HIGHER ORDER PROGRAMMING*

Raju Renjit. G,

rajurenjitgrover@yahoo.com, Grover house, Tripunithura, Eranakulam, Kerala, India.

Abstract

In the first section of this endeavor, we present set theoretic programming. And then state diagrams and declarative in the second and last.

*Thank You JESUS.

Contents

1 Set theoretic

Consider

- The
  - Packages:

$$\text{packageA}, \quad \text{packageB}, \quad \text{packageC} \quad \textit{and} \quad \text{packageD}.$$

- Then we see that,
  - The
    - ▷ View
      - From:

$$\text{packageB} \quad \textit{to}: \quad \text{packageA}$$

is

- Exactly
  - The
    - ▷ Same:
      - As

that

- From:

$$\text{packageC} \quad \textit{and} \quad \text{packageD}.$$

- But
  - In
    - ▷ Certain:
      - Cases,

some

- Classes

- Of:

packageA

should

- Be
  - Visible
    - ▷ In:

packageB,

- And
  - Invisible
    - ▷ In:

packageC *and* packageD.

- And so
  - We
    - ▷ Allow:
      - Packages

to

- Have:

private-classes, *protected-classes* *and* *public-classes*,

- And
  - Also
    - ▷ To:
      - Be extended.
- Exemplifying,

- ○ If:

```
package somePackage;

private class ClassOne{

    ⋮

}
```

- Then:

ClassOne

will

- Be
    - ○ Visible
        - ▷ Only
            - – In:

somePackage,

- And
    - ○ If:

```
package somePackage;

protected class ClassTwo{

    ⋮

}
```

- Then:

ClassTwo

will

- Be
  - Visible
    - ▹ Only
      - In:

somePackage,

- And
  - In:
    - ▹ Extended
      - Packages,
- And
  - If:

```
package somePackage;

public class ClassThree{

    ⋮

}
```

- Then:

ClassThree

will

- Be
    - Visible:
        - ▷ Everywhere,
- And
    - If:

subPackage *extends*: superPackage,

all

- Files
    - Of:

subPackage

should

- Start
    - With:

package subPackage extends superPackage;

- And
    - If:

SomeClass

belongs

- To:

superPackage,

we

- Can

- Write:

```
package subPackage extends superPackage;

protected class SomeClass extends super.SomeClass{

    ⋮

}
```

- And
  - We
    - Can
      - Write:

```
@("<superPackage-path>")
package subPackage extends superPackage;

@("/<dir-name>/<sub-dir-name>/")
import ...;

@("/<dir-name>/<sub-dir-name>/")
import ...;

⋮
```

to

- Specify
  - The
    - Path
      - Of:

superPackage.

- And
    - If:

subPackage *and* superPackage

have

- The
    - Same:
        - ▹ Paths,

we

- Write:

```
package subPackage extends superPackage;
```

- And we
    - Say
        - ▹ That:
            - Packages

cannot

- Contain
    - Interfaces,
        - ▹ But:
            - Only classes,
- And
    - Only
        - ▹ An:
            - Ipackage

can

- Contain
  - Interfaces,
    - And:
      - There

can

- Be:

private-interfaces, *protected-interfaces* and *public-interfaces*.

- Let:

```
ipackage  someIPackage;

⋮
```

- And
  - If:

```
package  somePackage;

⋮
```

then

- Any
  - Class
    - Of:

somePackage

can

- Implement
    - Any
        - ▹ Public-interface
            - Of:

someIPackage.

- But
    - If:

```
@("<superPackage-path>", "<someIPackage-path>")

package somePackage extends superPackage
                    implements someIPackage;

 // Note that, the first string corresponds to: superPackage

 // and the second to: someIPackage.

 // And if: somePackage extends: superPackage

 // and also implements: iPackageOne and iPackageTwo,

 // and: somePackage, iPackageOne and iPackageTwo

 // have the same paths, we write:

 // @("<superPackage-path>", , ).

⋮
```

then

- All
    - Public
        - ▷ Interfaces
            - Of:

someIPackage

should

- Be
    - Implemented
        - ▷ By:
            - At least

one

- Class
    - Of:

somePackage.

- And
    - Classes
        - ▷ That:
            - Does

not

- Belong
    - To
        - ▷ Any:
            - Package

can

- Extend
    - Any
        - ▹ Public-class:
            - Of

a

- Package
    - Or
        - ▹ Implement:
            - Any number

of

- Public
    - Interfaces
        - ▹ Of:
            - Ipackages.
- And
    - Similarly,
        - ▹ For:
            - Interfaces.
- And
    - If:

private class ClassOne{...} *and* protected class ClassTwo{...}

does

- Not
    - Belong:
        - ▹ To

any

- Package,
    - Then:

ClassOne *and* ClassTwo

will

- Be
    - Invisible
        - In:
            - Packages.
- And
    - Similarly,
        - For:
            - Interfaces.
- And
    - There
        - Should:
            - Be

exactly

- One:

class *or* *interface*

that

- Is:

private *or* protected *or* public

in

- All
    - Files.
        - ▹ And:
            - If

a

- Constructor
    - Does
        - ▹ Not:
            - Have

any

- Parameter,
    - Then
        - ▹ It:
            - is

the

- Default
    - Constructor.
        - ▹ And so
            - In:

```
class SomeClass{
    public SomeClass(){
        // This is the default constructor.
    }
    public SomeClass(int i){
        // This is not the default constructor.
    }
    ⋮
}
```

the

- First
  - ○ One:
    - ▷ Is

the

- Default
  - ○ Constructor.
    - ▷ And:
      - We

say

- That,
  - ○ All

▷ Classes:
  – Should

have

- A default
    - Constructor.
        ▷ And
          – If:

SomeClass obj1, obj2;

- Then:

obj1 *and* obj2

will

- Be
    - Initialized:
        ▷ Using

the

- Default
    - Constructor.
        ▷ Assume:
          – That,

the

- Default
    - Constructor
        ▷ Of:

SomeClass

is

- Invisible
    - In
        - ▹ The:
            - Environment,
- And
    - That:

SomeClass(int);

is

- Visible
    - In
        - ▹ The:
            - Environment.
- Then:

SomeClass sc;

will

- Not
    - Compile,
        - ▹ Since:

sc

could

- Not
    - Be
        - ▹ Initialized:

- With

the

- Default
  - Constructor.
    - ▷ And:
      - So

to

- Avoid
  - That:
    - ▷ Error,

we

- Write.

$$\text{SomeClass sc} = \text{null;}$$

or

$$\text{SomeClass sc} = \text{new SomeClass}(10);$$

- And
  - So
    - ▷ In:
      - General,

all

- Classes
  - Should:
    - ▷ Have

a

- Default
    - Constructor,
        - And:
            - If

we

- Declare
    - An:
        - Instance,

then

- It
    - Will
        - Automatically:
            - Be initialized

with

- The:
    - Default
        - Constructor,
- Or
    - We
        - Should:
            - Initialize it

with

- Some
    - Constructor

      ▹ Or:

null.

- And
  - So:

```
class SomeClass{
    public int i;
    public SomeClass next;
    public SomeClass(){}
}
```

should

- Be
  - Rewritten
      ▹ As:

```
class SomeClass{
    public int i;
    public SomeClass next = null;
    public SomeClass(){}
}
```

- And

- By
  - ▷ Default:

byte, short, int *and* long

will

- Be
  - Initialized
    - ▷ To:

0,

- And:

float	*to*:	0.0f,
double	*to*:	0.0,
boolean	*to*:	false,
char	*to*:	′a′,

- And
  - Arrays
    - ▷ To
      - Length:

0.

- And
  - Finally
    - ▷ Block:
      - Of

all

- Instances
  - Of:

```
class SomeClass{
    public SomeClass(){...}
    ⋮
    finally{
        // super; is optional.
        // And: super; will be ignored, if this block
        // is not present in the super class.
        ⋮
    }
}
```

in

- The
  - Memory
    - ▹ Will:
      - – Be executed

just

- Before

- ◦ The
  - ▹ Program:
    - – Halts

without

- Any
  - ◦ Specific:
    - ▹ Order.
- And
  - ◦ We
    - ▹ Can
      - – Write:

```
public  class  SomeClass{
    SomeClass{
        ⋮
    }
    ⋮
}
```

for

- Property
  - ◦ Blocks.
    - ▹ And
      - – Statements like:

$$\text{int} \quad i \quad = \quad \ldots;$$

cannot

- Be
  - Written
    - ▹ In:
      - Them.
- And
  - If:

```
public interface SomeInterface{
    SomeInterface{
        i            > 80;
        methodOne(i) > 800;
    }
    public int methodOne(int i);
    public void methodTwo(int i, int j);
}
```

then

- The
  - Value:
    - ▹ Of

the

- First
  - Parameter
    - ▷ Given:
      - To

the

- Implementations
  - Of:

$$\text{int} \quad \text{methodOne(int i);} \qquad (1)$$
$$\text{void} \quad \text{methodTwo(int i, int j);}$$

should

- Be
  - Greater
    - ▷ Than:

$$80,$$

- And
  - The
    - ▷ Value:
      - Returned

by

- The
  - Implementation
    - ▷ Of:
      - Method 1

should

- Be
  - Greater
    - ▹ Than:

800,

- Or
  - There
    - ▹ Will:
      - Be

an

- Exception.
  - And
    - ▹ If:

```
package  somePackage;

    somePackage{

        //  Later.;

    }

⋮
```

- Then:

// Later.

- And
  - Property

- ▹ Blocks:
  - Of packages

will

- Be
  - Applicable
    - ▹ Only:
      - In

the

- File.
  - And
    - ▹ If:

```
ipackage superIPackage;

    superIPackage{

        public class SomeClass implements InterfaceOne;

    }

⋮
```

then

- All
  - Packages
    - ▹ That
      - Implements:

superIPackage

should

- Have:

```
public class SomeClass implements InterfaceOne{
    ⋮
}
```

- And
  - If:

```
ipackage subIPackage extends superIPackage;
    subIPackage{
        public class SomeClass implements InterfaceTwo;
    }
⋮
```

- Then:

```
public class SomeClass implements InterfaceOne;
```

will

- Be
  - Overriden.
    - ▹ And

      - If:

// Later.

- And
    - Property
        - ▷ Block:
            - Of

an

- Ipackage
    - Is
        - ▷ The:
            - Union

of

- All
    - Property
        - ▷ Blocks:
            - In

all

- Files.
    - And
        - ▷ Conflicts:
            - Can

be

- Checked,
    - Since

- ▷ We:
  - Can check

whether

- Two
  - Sytnax-trees
    - ▷ Are:
      - The same.
- And
  - All
    - ▷ These:
      - Properties

should

- End
  - With
    - ▷ A semicolon.
      - Let.

```
public @interface AnnotationInterface{
    int i;
}
```

- Then
  - We
    - ▷ Can
      - Write:

```
public @class AnnotationClass{
    @int i;
    public AnnotationClass(){...}
    public AnnotationClass(@int i){...}
    public @AnnotationInterface annotationReturner(@int i){
        @int j                    =    i;
        @if (i < this.i)   j      =    this.i;
        return @AnnotationInterface(i = j);
        // We do not write:
        // return new @AnnotationInterface(i = j);
        // since we cannot create instances of
        // annotation classes and interfaces.
    }
}
```

- And
  - Then:

```
@AnnotationClass.annotationReturner(8)
void  someMethod(...){...}
```

- And
  - If
    - ▷ There:
      - Is ambiguity,

we

- Write:

```
(@AnnotationInterface)@AnnotationClass.annotationReturner(8)
```

- And
  - Since
    - ▷ The:
      - Values

in

- The
  - Fields
    - ▷ Of:

AnnotationClass

should

- Not
  - Vary:
    - ▷ During

the

- Entire
    - Compilation,
        - ▹ We:
            - Say

that,

- Values
    - In
        - ▹ The:
            - Fields

of

- Annotation
    - Classes
        - ▹ Can:
            - Only

be

- Changed
    - In
        - ▹ The:
            - Constructor.
- Or
    - Fields
        - ▹ Of:
            - Such classes

are

- Read
    - Only
        - ▹ In:
            - Methods.
- And
    - So
        - ▹ We
            - Write:

```
import  AnnotationClass;

@AnnotationClass(80)
class  ClassOne{

    ⋮

}

@AnnotationClass()
class  ClassTwo{

    ⋮

}
```

- And
    - The
        - ▹ Default-constructor
            - Of:

AnnotationClass

will

- Be
  - Used
    - ▷ In.

```
import AnnotationClass;

class ClassThree{

    public ClassThree(){}

    @AnnotationClass.annotationReturner(8)
    void someMethod(...){...}

}
```

- Let:

packageOne *and* packageTwo

be

- Extensions
  - Of
    - ▷ The
      - Package:

somePackage,

- And
  - Let:

ClassOne *and* ClassTwo

be

- Extensions
    - Of
        - ▹ The
            - Class:

$$\text{SomeClass.}$$

- The
    - Interpretation
        - ▹ Of:

$$\text{@switch}(\text{default},\quad \text{packageOne})$$

- Is:

“*use*: *default-values* *instead of the things of*: packageOne,”

- And
    - That
        - ▹ Of:

$$\text{@switch}(\text{packageTwo},\quad \text{packageOne}) \tag{2}$$

- Is:

“*if*: packageOne *and* packageTwo

have a common super package, and also have the same paths;

then use: packageTwo *instead of*: packageOne.”

- And
    - Expression 2
        - ▹ Will:

– Produce

an

- Exception,
    - If:

packageOne *and* packageTwo

does

- Not
    - Have
        - ▹ A common:
            - – Super-package.
- And
    - Similarly,
        - ▹ For:

@switch(ClassTwo, ClassOne)

- And
    - If:

```
public @class AnnClass{
    public AnnClass(){}
    private @boolean boolReturner(@int i){
        @boolean b = NonAnnotationClass.StaticMethod(i);
        // Note that, since the value of:
        // NonAnnotationClass.StaticMethod(i) is give to

        // a variable of type: @boolean,

        // we see that: StaticMethod of: NonAnnotationClass

        // will be invoked.

        @return b;

    }

    public @switch switchReturner(@import dt1,
                                  @import dt2,
                                  @int i){

        @if (@boolReturner(i)) @return @switch(dt1, dt2);

        @else @return @switch(,);    // Do nothing.

    }

}
```

we

- Can
  - Write:

```
    package somePackage;

    @("<path-of-AnnClass>", "<path-of-packageOne>")
    @AnnClass.switchReturner(packageTwo, packageOne, 8)
    import packageOne.*;

    // We did not write:
```

```
// @("<path-of-packageOne>", "<path-of-AnnClass>"),

// since we follow the order in which:

// AnnClass and packageOne

// is written.

⋮
```

- And
  - If:

```
@("<path-of-AnnClass>", "")
@AnnClass.switchReturner(default, ClassOne, 8)
import ClassOne;

public class ClassTwo{

    ClassOne [] arr;

    public ClassTwo(){

        arr          =   ...;

        int      i   =   arr[0].<some-method>();

    }

}
```

- And:

```
    @AnnClass.switchReturner(default, ClassOne, 8)
```

- Returns:

@switch(default, ClassOne),

- Then:

arr

will

- Only
  - Store:

null,

- And:

i == <default-value>.

- And
  - Since:

```
int sum(int [] arr){
    int result = 0;
    for(int i = 0; i<arr.length; i++){
        result += arr[i];
    }
    return result;
}
```

- And:

```
float sum(float [ ] arr){
    float result = 0.0f;
    for(int i = 0; i<arr.length; i++){
        result += arr[i];
    }
    return result;
}
```

are

- Similar,
    - We
        - Can
            - Write:

```
public @class AnnClass{

    public AnnClass(){}

    public @void codePaster(@import dt,
                             @native var,
                             @native c){

        dt result = c;

        for(int i = 0; i<var.length; i++){

            result += var[i];

        }

        return result;

    }

    public @void codePasterTwo(@import dt1 extends ...,
                                @import dt2 implements ...,
                                @native var){

        ⋮

    }

}
```

- And:

```
public class SomeClass{

    public SomeClass(){}
```

```
        public int sum(int [ ] arr){
            @AnnClass.codePaster(int, arr, 0)
        }
        public float sum(float [ ] arr){
            @AnnClass.codePaster(float, arr, 0.0f)
        }
        public string someMethod(){
            string s = . . . ;
            @AnnClass.codePasterTwo(. . . , . . . , s)
            // Note that, if we write: int i, j;
            // then: i and j will be natives of the environment.
            // And so:
            // @AnnClass.codePasterTwo(. . . , . . . , s)
            // will not compile, if: s is not defined
            // in the environment.
            return s;
        }
    }
```

- Or

- Since
  - There
    - Is no:

$$@\text{return} \quad \ldots;$$

- In:

$$@\text{void} \quad \text{codePaster}(@\text{import} \quad \text{dt}, \quad @\text{native} \quad \text{var}, \quad @\text{native} \quad \text{c})\{\ldots\} \tag{3}$$

we see that,

- When
  - The
    - Compiler
      - Compiles:

$$@\text{AnnClass.codePaster}(\text{int}, \quad \text{arr}, \quad 0) \tag{4}$$

it

- Will
  - Replace:

$$\text{dt}, \qquad \text{var} \qquad \textit{and} \qquad \text{c}$$

in

- Method 3
  - With:

$$\text{int}, \qquad \text{arr} \qquad \textit{and} \qquad 0,$$

- And
  - Then:
    - Paste

the

- Resulting
  - Code
    - ▹ At:
      - The place

where

- Expression 4
  - Is:
    - ▹ Written.
- And
  - So
    - ▹ If:

```
public @class AnnClass{
    public @int i;
    public AnnClass(){}
    public AnnClass(@int i){
        this.i = i;
    }
    public @void codePaster(@native var, @native c){
        // We cannot write:
        // int i;
        // since: i is a field of this class,
        // and annotation variables have more preference.
        // But we can write:
        // @int i = this.i;
        // and annotation variables cannot be declared
        // in non annotation classes.
        // And annotation and non annotation variables
        // cannot be mixed in code that will be pasted.
        @if (i < 8) return var; @else return var + c;
    }
}
```

- Then:

```
@AnnClass(8)
public class SomeClass{
    public SomeClass(){}
    public int intReturner(int i){
        @AnnClass.codePaster(i, 1)
    }
    public string stringReturner(string s){
        @AnnClass.codePaster(s, "abc")
        }
        public string stringReturnerTwo(string s){
            @AnnClass.codePaster(s, intReturner(0) + "a")
        }
    }
```

will

- Be
    - Converted
        - To.

```
    public class SomeClass{
        public SomeClass(){}
        public int intReturner(int i){
            return i + 1;
        }
        public string stringReturner(int s){
            return s + "abc";
        }
        public string stringReturnerTwo(int s){
            return s + intReturner(0) + "a";
        }
    }
```

- Let:

```
    public @class AnnClass{
        public AnnClass(){}
        public @void codePaster(@native v1, @native v2){...}
    }
```

- And
    - We
        - Compile:

```
    class SomeClass{
        public SomeClass(){}
        public int someMethod(int i){
            int result = ...;
            @AnnClass.codePaster(i, result)
            return result;
        }
    }
```

we see that,

- Two
  - Variables
    - In:

$$@void \quad codePaster(@native, \quad @native); \qquad (5)$$

may

- Have
  - Also
    - Been
      - Named:

i *and* result.

- And so
  - When
    - We
      - Compile:

SomeClass,

all

- Variables
  - In
    - Method 5:
      - Will

be

- Renamed
  - Until
    - There:

      - Is

no

- Conflict.
  - ○ And
    - ▹ We
      - Can write:

```
protected @void header(@native v){...}
protected @void footer(@native v){...}
private @void generalCode(@native v1, @native v2){...}
public @void publicCodePaster(@native v){
    @header(v);
    @generalCode(v, "abc");
    // We do not allow:
    // @string s = ...;
    // @generalCode(v, s);
    @footer(v);
}
```

- And
  - If:

```
@interface AnnInterface{
    string someString();
    @AnnotationInterface annotationReturner(@int i);
}
```

- And

- If:

```
@class AnnClass implements @AnnInterface{
    ⋮
}
```

- Then:

AnnClass

should

- Implement
  - All
    - ▹ Methods
      - Of:

AnnInterface

that

- Returns
  - An:
    - ▹ Annotation.
- And
  - So:

AnnClass

should

- Implement:

@AnnotationInterface annotationReturner(@int i);

- And
  - If:

```
@AnnotationClass(...)
public class SomeClass{
    ⋮
}
```

- Then:

SomeClass@AnnotationClass == true,

- And
  - All
    - ▹ Public-methods
      - – Of:

AnnotationClass

that

- Returns
  - An:
    - ▹ Annotation
      - – Interface

should

- ○ ▹ –

// Later.

- And:

static

witten

- In
  - These:
    - ▷ Classes
      - And interfaces

will

- Be
  - Ignored.
    - ▷ And
      - If:

```
@ipackage annotationIPackage;

⋮
```

then

- We
  - Cannot
    - ▷ Write:

```
package somePackage implements @annotationIPackage;
```

- But
  - We
    - ▷ Can

– Write:

```
@package annotationPackage implements @annotationIPackage;
⋮
```

- And
  - We
    - ▹ Can:
      - – Give

a

- Description
  - For:

@package *and* @ipackage

like

- That
  - Which
    - ▹ We
      - – Did for:

packages and ipackages.

- And
  - If:

```
@package someAnnotationPackage;
⋮
```

- And
  - If:

```
@someAnnotationPackage
package  somePackage;

⋮
```

then

- All
  - Public-classes
    - ▹ Of:

someAnnotationPackage

should

- ● ○ ▹ –

// Later.

- The
  - Interpretation
    - ▹ Of:

$$\text{boolean} \quad b \quad = \quad \text{somePackage;} \tag{6}$$

- Is:

"*is*: somePackage *available*?"

- And
  - If

        - ▹ There:
            - Is ambiguity,

we

- Write:

$$\text{boolean} \quad b \quad = \quad (\text{package})\text{somePackage};$$

- And
    - Similarly:

$$\text{boolean} \quad b \quad = \quad \text{SomeClass};$$

is

- Equivalent
    - To.

$$\text{boolean} \quad b \quad = \quad (\text{class})\text{SomeClass};$$

- Let:

$$\text{void} \quad \text{methodForSomePutProperty}(\text{int}); \tag{7}$$

be

- Some
    - Method.
        - ▹ Then:
            - If

we

- Write:

$$\text{void} \quad \text{methodForSomePutProperty}(\text{int}) \quad \text{for} \quad \text{somePutProperty};$$

in

- The
    - Class
        - ▷ Body:
            - – The compiler

can

- Note
    - That,
        - ▷ Since:
            - – Method 7

has

- Only
    - One:
        - ▷ Parameter,
- And
    - Returns
        - ▷ Nothing:
            - – It

can

- Be
    - Used
        - ▷ To:
            - – Define

the

- Put
    - Property:

somePutProperty,

- And
  - So
    - ▷ Define:
      - It.
- And
  - So
    - ▷ If:

```
class SomeClass{
    SomeClass methodForPlusPlus(SomeClass) for ++;
    float methodForGetProperty() for someProperty;
    float methodForGetProperty(int) for someProperty;
    void methodForPutProperty(float) for someProperty;
    void methodForPutProperty(int, float) for someProperty;
    boolean methodForIsProperty() for someProperty?;
    boolean methodForIsProperty(int) for someProperty?;
    public SomeClass(){...}
    protected static void methodForPlusPlus(SomeClass sc){...}
    ⋮
}
```

we

- Can
  - Write.

```
SomeClass    sc          =        ...;
float        f           =        sc.someProperty;
f                        =        sc.someProperty[0];
sc.someProperty          =        0.0f;
sc.someProperty[0]       =        0.0f;
boolean      b           =        sc.someProperty?;
b                        =        sc.someProperty[0]?;
sc++;
```

- Note that,
  - If
    - ▷ We
      - Write:

```
void methodForSomeProperty(int, float) for someProperty;
```

the

- First
  - Parameter
    - ▷ Of:

```
void methodForSomeProperty(int, float);
```

should

- Be
    - Of
        - Type:

int.

- And:

++

- And
    - The
        - Get
            - Property:

someGetProperty

- And
    - A constructor
        - Of
            - Type:

(int, int)

should

- Be
    - Defined
        - In:
            - Classes

that

- Implements.

```
interface SomeInterface{
    (this|this)    for    ++;
    (int|)         for    someGetProperty;
    this(int i, int j);
    ⋮
}
```

- Let:

$$\text{obj}, \qquad \text{obj0}, \qquad \text{obj1}, \qquad \ldots \tag{8}$$

be

- Instances
  - Of:

$$\text{SomeClass},$$

- And
  - Let:

$$\text{j}$$

be

- Of
  - Type:

$$\text{int},$$

- And
    - We
        - Execute:

$$\text{int} \quad \text{i} \quad = \quad \text{obj?;}$$

- Then:

$$\text{i}$$

will

- Be
    - Equal
        - To:

$$\begin{array}{ccccc} 0, & 1, & 2, & 3, & 4, \\ 5, & 6, & 7 & \textit{or} & 8 \end{array}$$

- If:

$$\text{obj}$$

is

- Equal to null,
- Or unlocked,
- Or read-locked,
- Or write-locked,
- Or unlocked and not fully initialized,
- Or read-locked and not fully initialized,

- Or write-locked and not fully initialized,
- Or busy and unlocked,
- Or idle and unlocked,
    - ◦ Respectively.
        - ▷ And
            - – If:

$$\text{int} \quad \text{i} \quad = \quad (\text{obj0}, \quad \text{obj1}, \quad \ldots, \quad \text{j})?;$$

- Then:

$$\text{i} \;==\; \text{obj0}?,$$

if

- The
    - ◦ Values
        - ▷ Of:

$$\text{obj0}?, \qquad \text{obj1}?, \qquad \ldots \qquad \textit{and} \qquad \text{j}?$$

are

- The
    - ◦ Same,
        - ▷ And:

$$\text{i} \;==\; -1,$$

if

- The
    - ◦ Values
        - ▷ Of:

$$\text{obj0?}, \quad \text{obj1?}, \quad \ldots \quad \textit{and} \quad \text{j?}$$

are

- Different.
  - And
    - If:

$$\text{i} == -1,$$

- Then:

$$(\text{obj0}, \quad \text{obj1}, \quad \ldots \quad \text{j})?[1] == \text{true},$$

if

- Some
  - Objects
    - Of:
      - The list 8
- And
  - Not
    - All:
      - Of them

are

- Equal
  - To:

$$\text{null}.$$

- And
  - Similarly,
    - Using:

[2], [3], . . . [9].

- And
  - If:

```
class ClassOne{
    this.class for ClassTwo, ClassThree;
    ⋮
}
```

- Then:

ClassTwo *and* ClassThree

can

- Access
  - All
    - ▷ Protected-members
      - Of:

ClassOne.

- And
  - If
    - ▷ We
      - Write:

```
this.class.∗ for ClassTwo, ClassThree;
```

then

- Those
  - Classes
    - ▹ Can:
      - Access

all

- Members
  - Of:

ClassOne.

- And
  - There
    - ▹ Can:
      - Be

at

- The
  - Most
    - ▹ One:
      - Such statement,
- And
  - It
    - ▹ Cannot
      - Be:

static *or* final *or* public *or* protected *or* private.

- And
  - We
    - ▹ Can

    - Write:

```
package somePackage;
this.package for packageA, packageB;
// Or: this.package.* for packageA, packageB;
import ...;
⋮
```

- And
    - In:

```
class SuperClass{
    public SuperClass(){}
    public partial int intReturner(int i){...}
}
```

- Since:

int intReturner(int);

is

- Partial,
    - We see that:

SuperClass

is

- A partial
    - Class.
        - ▹ And:
            - – So

we

- Cannot
    - Create
        - ▹ An:
            - – Instance

of

- It.
    - Or:
        - ▹ Partial
            - – Classes

should

- Be extended,
    - And
        - ▹ The:
            - – Implementation

of

- All
    - Partial
        - ▹ Methods:
            - – In them

should

- Be
  - Completed.
    - ▹ And:
      - – So

to

- Complete
  - The
    - ▹ Implementation
      - – Of:

```
int intReturner(int);
```

we

- Can
  - Write:

```
class SubClass extends SuperClass{
    public SubClass(){}
    public int intReturner(int i){
        int j = 10;
        return super(i) + j;
    }
}
```

- Or:

```
class SubClass extends SuperClass{
    public SubClass(){}
    public int intReturner(int i){
        // super(int) is not called.
        return 10;
    }
}
```

- And
    - If:

SubClass

does

- Not
    - Override:

int intReturner(int);

then

- That
    - Method
        - ▹ Of:

SuperClass

will

- Be copied
    - As
        - ▹ Such
            - Into:

SubClass.

- And
    - So:

SubClass

will

- Still
    - Be:
        - ▹ Partial.
- And
    - Partial
        - ▹ Methods:
            - Can

be

- Extended
    - Without
        - ▹ Completing:
            - Its implementation.
- Exemplifying,
    - In:

```
class SubClass extends SuperClass{
    public SubClass(){}
    public partial int intReturner(int i){...}
}
```

the

- Partial
    - Method
        - Of:

SuperClass

is

- Extended,
    - But
        - Not:
            - Completed.
- And
    - So:

SubClass

will

- Still
    - Be:
        - Partial.
- And

        ◦ Partial
            ▹ Classes:
                – Should

have

- A default
    - ◦ Constructor.
        - ▹ And
            - – If:

partial SuperClass sc; (9)

then

- Even
    - ◦ Though:

SuperClass

maybe

- Non
    - ◦ Partial:

sc

cannot

- Point
    - ◦ To
        - ▹ An
            - – Instance of:

SuperClass,

- But

- Only
  - ▹ To:
    - An instance

of

- A subclass.
  - And
    - ▹ So:

sc

in

- Statement 9
  - Should
    - ▹ Be
      - Initialized with:

null,

- Or
  - With
    - ▹ An:
      - Instance

of

- A subclass.
  - And:

partial int i;

is

- Equivalent

    - To.

```
int i;
```

- And
    - So:

```
class SomeClass{
    public partial int i;
    public SomeClass(){}
}
```

is

- Not
    - Partial.
        - ▷ And:
            - Partial-interfaces

cannot

- Be
    - Implemented,
        - ▷ But:
            - Should

be

- Extended.
    - And:

```
partial package somePackage;
```

is

- Equivalent
  - To:

```
package somePackage;
```

- And
  - Similarly,
    - ▷ For:
      - Others.
- And:

```
abstract int someMethod();
```

can

- Be
  - Rewritten
    - ▷ As:

```
partial int someMethod(){ return 0; }
```

- And:

```
abstract class SomeClass{
    ⋮
}
```

- As:

$$\text{partial class SomeClass}\{$$
$$\vdots$$
$$\}$$

- And
  - If:

$$\text{volatile boolean boolReturnerTwo(int i)}\{\ldots\} \tag{10}$$

then

- Method 10
  - Can
    - ▷ Only:
      - Read

the

- Fields
  - Of
    - ▷ The:
      - Class.
- And
  - So
    - ▷ It:
      - Can

only

- Invoke
  - Volatile

        ▷ Methods:
            – Of

the

- Class,
    - Fields
        ▷ And:
            – Parameters.
- And
    - If
        ▷ We:
            – Override

a

- Volatile
    - Method,
        ▷ The:
            – New method

will

- Still
    - Be:
        ▷ Volatile.
- And
    - So
        ▷ The
            – Keyword:

volatile

should

- Be
  - Used
    - ▷ Appropriately.
      - And:

public class.test TestClass{

⋮

}

- And:

int.test testMethod(int i){…}

- Is:

a test-class *and* *a test-method*

respectively.

- And
  - Test
    - ▷ Methods:
      - Can

be

- Written
  - In:
    - ▷ Non
      - Test-classes.

- And
    - Since
        - ▷ The:
            - – Compiler

will

- Remove
    - Them
        - ▷ During:
            - – Optimization,

we

- Say
    - That,
        - ▷ Values:
            - – Returned

by

- Test
    - Methods,
        - ▷ And:
            - – Values stored

in

- Test
    - Variables
        - ▷ Can:
            - – Only

be

- Given
  - To:
    - ▹ Test
      - Variables,
- And
  - Test
    - ▹ Methods:
      - Cannot change

the

- Value
  - Of:
    - ▹ Non
      - Tests.
- And so
  - We
    - ▹ Can
      - Write:

```
int        j   =   8;

int.test   i   =   testMethod(j);
```

- But
  - Not:

```
int   j   =   testMethod();
```

- And

- Non
    - ▹ Test:
        - Variables

are

- Read
    - Only
        - ▹ In:
            - Test-methods,
- And
    - Test
        - ▹ Methods:
            - Can

only

- Access
    - Volatile
        - ▹ And:
            - Test-methods

of

- The:
    - Class,
        - ▹ Fields
            - And parameters.
- And
    - If:

```
class NonTestClass{
    ⋮
}
class.test TestClass extends NonTestClass{
    ⋮
}
```

then

- We
    - Cannot
        - ▷ Write:

          ```
          NonTestClass obj = new TestClass();
          ```
- And
    - If:

```
package superPackage;
public class SomeClass{
    public SomeClass(){...}
    ⋮
}
```

we

- Can
  - Write:

```
package subPackage extends superPackage;
public class.test DifferentName extends super.SomeClass{
    public DifferentName(){...}
    ⋮
}
```

- But
  - Not:

```
package subPackage extends superPackage;
public class.test SomeClass extends super.SomeClass{
    public SomeClass(){...}
    ⋮
}
```

- And
  - Similarly,
    - ▹ For:
      - Methods.

- And
  - If:

final package someFinalPackage;

we

- Can
  - Write:

package.test testPackage extends someFinalPackage;

- And
  - All
    - ▷ Classes
      - Of:

testPackage

will

- Be
  - Tests.
    - ▷ And:
      - Similarly,

for

- Classes.
  - And
    - ▷ If:
      - We override

a

- Test,

    - The:
        - New
            - One

will

- Also
    - Be:
        - A test.
- And
    - So
        - The:
            - Test-modifier

should

- Be
    - Used:
        - When

we

- Override
    - Tests.
        - And:

partial *and* final

cannot

- Be
    - Used
        - With:
            - Tests.

- And
    - Classes
        - ▹ That:
            - Does

not

- Extend
    - Any
        - ▹ Other:
            - Class

will

- Extend:

```
public class DefaultSuperClass{
    public DefaultSuperClass(){}
    public void.test print(int i){...}
    ⋮
}
```

by

- Default.
    - And:

DefaultSuperClass

will

- Not
    - Extend:
        - ▹ Itself.
- And
    - We
        - ▹ Do not
            - Allow:

```
@class.test   AnnotationClass{
    ⋮
}
```

- And:

```
interface   SomeInterface{
    ⋮
}
```

can

- Be
    - Rewritten
        - ▹ As:

$$\text{iclass SomeInterface}\{$$

$$\vdots$$

$$\}$$

- And:

$$\text{static final int i} = 0123;$$

- As:

$$\text{final final int i} = (4)123;$$

- And
    - Similarly,
        - For:
            - All

other

- Integers
    - Between:

$$1 \quad \textit{and} \quad 17.$$

- And
    - We
        - Can
            - Write:

$$(\text{null})\text{int i, j;} \quad \textit{and} \quad (+)\text{int i, j;}$$

for

- Unsigned

    - Integer
        - And nullables:
            - Respectively.

- And:

$$(+)(\text{null})\text{int} \quad \text{i};$$

is

- Equivalent
    - To.

$$(\text{null})(+)\text{int} \quad \text{i};$$

1.1 Lists

Let.

$$\text{int} \;\; \text{boolean} \quad \text{ib};$$

- Then:

$$\text{ib}, \qquad \text{ib}[0] \qquad \textit{and} \qquad \text{ib}[1]$$

will

- Be
    - Of
        - Type:

$$\text{int} \;\; \text{boolean}, \qquad \text{int} \qquad \textit{and} \qquad \text{boolean}$$

$$\textit{respectively}.$$

- And
    - So
        - If:

$$\text{ib} = 8\ \text{true};$$

- Then:

$$\text{ib}[0] == 8 \qquad \textit{and} \qquad \text{ib}[1] == \text{true}.$$

- And
  - We
    - ▹ Can
      - Write.

$$\begin{aligned}
&\text{int}\ \ \text{int}\ \ \text{t} &&= 10\ \ 20, \quad \text{t2} = 30\ \ 40;\\
&\text{int}\ \ \text{i} &&= \text{t}[0];\\
&\text{t}[0] &&= 50;\\
&\text{t} &&= \text{i}\ \ \text{i};\\
&\text{boolean}\ \ \text{b} &&= \text{t} == 10\ \ 10\ \| \ \text{t} == \text{i}\ \ \text{i}\ \|\ \text{t} == \text{t2};
\end{aligned}$$

- Let:

$$\text{i} \qquad \textit{and} \qquad \text{j}$$

be

- Of
  - Type:

$$\text{int},$$

- And
  - Let:

t1, t2 *and* t3

be

- Of
  - Type:

int int,

- And
  - Let:

t5

be

- Of
  - Type:

byte int float char.

- Then
  - The
    - ▹ Type
      - Of:

$t5[2, \ 0]$

- Is:

float byte,

- And
  - That
    - ▹ Of:

$t5[2, \ 0, \ 3]$

- Is:

$$\text{float} \quad \text{byte} \quad \text{char.}$$

- And
  - So:

$$\begin{aligned} \text{int} \quad \text{int} \qquad\qquad & \text{t1} & = & \quad \ldots; \\ \text{int} \quad \text{float} \qquad\qquad & \text{t6} & = & \quad \ldots; \\ \text{int} \quad \text{float} \quad \text{int} \quad \text{int} \quad & \text{t7} & = & \quad \text{t1}[0] \quad \text{t6}[1] \quad \text{t6}[0] \quad \text{t1}[1]; \end{aligned}$$

can

- Be
  - Rewritten
    - As:

$$\begin{aligned} \text{int} \quad \text{int} \qquad\qquad & \text{t1} & = & \quad \ldots; \\ \text{int} \quad \text{float} \qquad\qquad & \text{t6} & = & \quad \ldots; \\ \text{int} \quad \text{float} \quad \text{int} \quad \text{int} \quad & \text{t7} & = & \quad \text{t1}[0] \quad \text{t6}[1, \quad 0] \quad \text{t1}[1]; \end{aligned}$$

- And:

$$\text{int} \quad \text{int} \quad \text{int} \quad \text{int} \quad \text{int} \quad \text{int} \qquad \text{t4} \qquad = \qquad \text{t2}[0] \quad \text{t2}[1] \quad \text{t3}[0] \quad \text{t3}[1] \quad \text{t3}[0] \quad \text{i};$$

- As.

$$\text{int} \quad \text{int} \quad \text{int} \quad \text{int} \quad \text{int} \quad \text{int} \qquad \text{t4} \qquad = \qquad \text{t2} \quad \text{t3} \quad \text{t3}[0] \quad \text{i};$$

- The

- Interpretation
    - ▹ Of:

$$\mathrm{t1} \quad = \quad \mathrm{t2} \; + \; 10 \;\; 10;$$

- Is:

$$\mathrm{t1}[0] \quad = \quad \mathrm{t2}[0] \; + \; 10;$$

$$\mathrm{t1}[1] \quad = \quad \mathrm{t2}[1] \; + \; 10;$$

- And
    - Similarly,
        - ▹ For:
            - Other operators.
- Let:

$$\mathrm{t8}$$

be

- Of
    - Type:

$$\mathrm{int} \;\; \mathrm{int} \;\; \mathrm{int} \;\; \mathrm{int} \;\; \mathrm{int}.$$

- Then
    - We see that,
        - ▹ Even
            - Though:

$$\mathrm{t8} \quad = \quad 10 \; + \; 10 \;\; 20 \;\; 30 \; + \; 30 \;\; 40 \;\; 50 \; + \; 50; \qquad (11)$$

is

- Parsable,
    - It
        - Is:
            - Confusing.
- And
    - So
        - We:
            - Say

that,

- Inside
    - Tuples
        - With:
            - More

than

- One
    - Location:
        - Operations
            - Involving

lesser

- Number
    - Of
        - Locations:
            - Should

be

- Enclosed

- In
  - Between:

(*and*),

- And
  - In
    - Operations:
      - Involving tuples

with

- More
  - Than
    - One locations:
      - Operands

with

- Lesser
  - Number
    - Of:
      - Locations

should

- Be
  - Enclosed
    - In
      - Between:

(*and*).

- And so
  - Statements

▹ Like:
  – Statement 11

should

- Be
  - Rewritten
    ▹ As:

$$t8 = (10 + 10)\ 20\ (30 + 30)\ 40\ (50 + 50);$$

- And:

$$t1 = 10\ 10 + 20\ 20;$$

$$t1 = t2 + 20\ 20;$$

$$t1 = t2\ /\ 10\ (10 + 10) + t1 * 10\ (10 + 10);$$

- As:

$$t1 = (10\ 10) + (20\ 20);$$

$$t1 = t2 + (20\ 20);$$

$$t1 = t2\ /\ (10\ (10 + 10)) + t1 * (10\ (10 + 10));$$

- But:

$$\text{int}\ \ i = 10 + 20 * 20 + 30 * 30;$$

need

- Not

    - Be:
        - ▷ Rewritten.

- And
    - We
        - ▷ Can
            - Write:

```
int   int [ ]     arr    =     new   int   int[8];
```

- But
    - Since:

```
int   int   [ ]   int     t9;
```

will

- Complicate,
    - We
        - ▷ Do:
            - Not

allow

- Arrays
    - In:
        - ▷ Tuples.

- And
    - We
        - ▷ Say
            - That:

```
(int   [ ]   arr)
```

is

- Equivalent
    - To:

```
class <class-name-hidden-from-programmers>{
    public int [ ] arr;
    public <class-name-hidden-from-programmers>(){
        this.arr = new int[0];
    }
    public <class-name-hidden-from-programmers>(int [ ] arr){
        this.arr = arr;
    }
}
```

- And
    - We
        - ▹ Can
            - – Write:

```
int (int i) int  t12  =  8 null 9;

int (int i) int  t13  =  10 new(11) 12;

t12                   =  t13;

t12[1].i              =  t12[0] – t12[1].i;

// We did not write: (int i;),

// since we do not write: for(...;  ...;  i++;){...}
```

- But
  - Not:

```
(int i)  pc1  =  ...;

(int j)  pc2  =  pc1;
```

- And
  - We
    - ▹ Do:
      - Not

allow

- Inner
  - Pseudoclasses.
    - ▹ And:
      - There

can

- Be

        ◦ Only
            ▹ One:
                – Field

in

- Them.
    ◦ And
        ▹ So:
            – We

we

- Do
    ◦ Not
        ▹ Allow:

$$((\ldots)\quad f) \qquad \textit{and} \qquad (\text{int}\quad i,\quad j).$$

- And
    ◦ We
        ▹ Allow
            – Methods like.

(int [ ] arr) someMethod((int [ ] arr));

- Let:

int int tupleReturner(int);

be

- Some
    ◦ Method.
        ▹ Then:

$$i \;\; j \quad = \quad \text{tupleReturner(i);}$$

is

- Equivalent
    - To:

```
int  int    t   =   tupleReturner(i);
i               =   t[0];
j               =   t[1];
```

- And:

$$i \;\; i \quad = \quad \text{tupleReturner(i);}$$

- To:

```
int  int    t   =   tupleReturner(i);
i               =   t[1];
```

- And:

```
int           i0   =   0,   i1   =   1;
fint  float   t6;
int           i    =   t6[i0];
float         f    =   t6[i1];
```

should

- Be
  - Rewritten
    - ▷ As:

```
int  float    t6;

int           i     =    t6[0];

float         f     =    t6[1];
```

- And
  - The
    - ▷ Value
      - Of:

$$(\text{obj0}, \quad \text{obj1}, \quad \text{obj2}, \quad \ldots \quad)?$$

can

- Be
  - Given
    - ▷ To:
      - Tuples

of

- Width:

$$10.$$

- And
  - If:

$$[\text{int}\ \ \text{int}]\quad \text{k};$$

- Then:

$$\text{k}$$

will

- Be
  - A list
    - Of:

$$\text{int}\ \ \text{int},$$

- And:

$$\text{k.length} \ == \ 0.$$

- But
  - If:

$$[\text{int}\ \ \text{int}]\quad \text{k}\quad = \quad 10\ \ 10, \quad 20\ \ 20, \quad 30\ \ 30; \tag{12}$$

- Then:

$$\text{k.length} \ == \ 3.$$

- And
  - So
    - We:
      - Do

not

- Use:

$$\text{new}$$

to

- Initialize
    - Lists.
        - And:
            - List elements

can

- Be
    - Accessed
        - Similar:
            - To

that

- Of
    - Arrays.
        - And:
            - So

after

- Executing
    - Statement 12:

$$k[0] \quad == \quad 10 \;\; 10 \qquad \textit{and} \qquad k[1] \quad == \quad 20 \;\; 20.$$

- And
    - We
        - Can
            - Write:

```
[int  int]        k2   =   10  10,     20  20,     30  30,     40  40;
[int  int  int]   k3   =   10  10  10,     20  20  20;
if  (k2[3][0]  ==  40)       k2[3][0]   =   50;
if  (k3[1]  ==  20  20  20)  k3[1]      =   80  80  80;
k2                     =   ;  //  Remove  all  elements.
                              //  We  do  not  use:  null,
                              //  since  we  did  not  use:  new.
```

- And
  - Since
    - ▷ We:
      - Do

not

- Allow:

```
int  int  [ ]  int    t;
```

we

- Do
  - Not
    - ▷ Allow:
      - Arrays

in

- Lists.
  - But

      ▷ We:
         – Can

have

- Arrays
    - Of
        ▷ List.
           – Exemplifying:

```
[int  int]       k1    =    ...;

[int  int]       k2    =    ...;

[int  int]  [ ]  arr   =    (  k1,  k2  );
```

- And
    - To
        ▷ Avoid:
           – Congestion,

we

- Say
    - That:

```
[int  int]  [ ]    arr    =    (

                                   (  10  10,     20  20,     30  30  ),

                                   (  40  40,     50  50,     60  60  ),

                               );
```

should

- Be
  - Rewritten
    - ▷ As:

```
[int  int]       k1   =   10  10,    20  20,    30  30;
[int  int]       k2   =   40  40,    50  50,    60  60;
[int  int]  [ ]  arr  =   (  k1,  k2  );
```

- And
  - To
    - ▷ Avoid:
      - Complications,

we

- Do
  - Not
    - ▷ Allow:
      - Inner-lists.
- And
  - So
    - ▷ We:
      - Do

not

- Allow
  - Statements

▷ Like.

```
[int]          k5     =    ((10,  10),  20,  20),  40,  40;
[[int]]        k29    =          (10,  10),    (20,  20);
[int  [int]]   k30    =      9  (10,  10),    19  (20,  20);
```

- Let:

$$\text{k} \qquad \textit{and} \qquad \text{k1}$$

be

- Instances
  - Of:

$$[\text{int} \quad \text{int}],$$

- And
  - Let:

$$\text{i} \qquad \textit{and} \qquad \text{j}$$

be

- Of
  - Type:

$$\text{int}.$$

- Then
  - In:

$$\text{k} \quad = \quad 10 \;\; 10, \quad 20 \;\; 20;$$

we

- Say
  - That:

$$10 \quad 10, \qquad 20 \quad 20$$

is

- A list
  - Literal.
    - ▹ And similarly,
      - In:

$$k \quad = \quad i \quad i, \quad j \quad j;$$

- And
  - If:

$$k \quad = \quad 10 \quad 10, \quad k1, \quad 20 \quad 20, \quad 30 \quad 30; \tag{13}$$

we see that,

- Since:

$$k \qquad \textit{and} \qquad k1$$

are

- Pointers,
  - We:
    - ▹ Will

be

- Forced
  - To
    - ▹ Append:

20 20 *and* 30 30

- To:

k1.

- And
    - So
        - ▹ We:
            - Say

that,

- If
    - A commas
        - ▹ Separates:
            - A list

from

- An already
    - Declared:
        - ▹ Variable
            - Or literal,

a

- Copy
    - Of
        - ▹ That:
            - List

will

- Be
    - Made.

▹ And:
  - So

when

- We
  - Execute:
    - ▹ Statement 13

a

- Copy
  - Of:

$$k1$$

will

- Be
  - Used.
    - ▹ But:
      - If

we

- Write:

$$k = k1; \tag{14}$$

we see that,

- Since
  - No
    - ▹ Comma
      - Separates:

$$k1$$

from

- Any already
  - Declared
    - Variable:
      - Or literal,

no

- Copy
  - Of:

k1

will

- Be
  - Made.
    - And:
      - So

if

- We
  - Execute:
    - Statement 14,

the

- Address
  - In:

k1

will

- Be

- Given
  - To:

k.

- And
  - We
    - Can
      - Say that:

$k \;=\; 10 \;\; 10, \;\; 20 \;\; 20;$ *and* $k \;=\; (10 \;\; 10, \;\; 20 \;\; 20);$

are

- Equivalent,
  - So that,
    - We
      - Can write.

[int int] $k \;=\; (10 \;\; 10, \;\; 20 \;\; 20),$ k2;

- But
  - In:

[int int] $k \;=\; 10 \;\; 10, \;\; 20 \;\; 20,$ k2;

we see that,

- If:

k2

has

- Not
  - Been

      ▷ Declared:
        – Earlier,

the

- Compiler
    - Can
      ▷ Recognize:
        – It

as

- A list
    - Declared
      ▷ Along
        – With:

k,

- And
    - If:

k2

has

- Been
    - Already
      ▷ Declared:
        – As

a

- Variable
    - Of
      ▷ Type:

int int,

the

- Compiler
  - Can
    - ▹ Understand
      - – That:

k2

is

- Used
  - To
    - ▹ Initialize:

k,

- And
  - If:

k2

has

- Been
  - Already
    - ▹ Declared:
      - – As

a

- List
  - Of
    - ▹ Type:

[int int],

the

- Compiler
    - Can
        - ▷ Note:
            - That

a

- Comma
    - Separates:

k2

from

- A literal.
    - And
        - ▷ So:
            - Append

a

- Copy
    - Of
        - ▷ It:
            - To

the

- End
    - Of:

k.

- And
    - In:

      $[\text{int int}]\quad k2 = k1,\quad k20 = 10\ 10,\quad 20\ 20;$

we see that,

- Since
    - No
        - ▹ Variable
            - Named:

$$k20$$

has

- Been
    - Declared
        - ▹ So:
            - Far,
- And
    - Since
        - ▹ We:
            - Had

stated

- That:

$$k1$$

is

- Of
    - Type:

[int   int],

- And
    - Since
        - ▹ No comma
            - Separates:

k1

from

- Any already
    - Declared
        - ▹ Variable:
            - Or literal,

the

- Address
    - Of:

k1

will

- Be
    - Given
        - ▹ To:

k2.

- And so
    - We
        - ▹ Can

      - Write.

```
int i = ...;
// No variable named: k7 or k8 has been declared so far.
[int] k5 = ...;
[int] k6 = 10, 20, k5, i, k7, k8 = 40, 50;
```

- Let:

k2

be

- An
  - Instance
    - Of:

[int int].

- Then
  - If
    - We
      - Exceute:

```
k1 += k2;
```

a

- Copy
  - Of:

k2

will

- Be
  - Appended
    - To:

$$\text{k1}.$$

- And
  - We
    - Can
      - Write:

$$\text{k1} \quad += \quad 10 \;\; 10, \quad 20 \;\; 20, \quad \text{k2}, \quad 30 \;\; 30, \quad 40 \;\; 40;$$

- And
  - If:

$$\text{k.length} - 1 < \text{i},$$

- Then:

$$\text{k[i]}$$

will

- Be
  - Equivalent
    - To:

$$\text{k[k.length} - 1\text{]}.$$

- And
  - Similarly,
    - If:

$$i \quad < \quad 0,$$

- Then:

$$k[i]$$

will

- Be
    - Equivalent
        - To:

$$k[0].$$

- And
    - If:

$$k.length \quad == \quad 0,$$

- Then:

$$k[8] \quad = \quad \ldots; \qquad \textit{and} \qquad k[8] \quad += \quad \ldots;$$

will

- Be
    - Equivalent
        - To:

$$k \quad += \quad <\text{default-value}> \quad <\text{default-value}>;$$

$$k \quad = \quad \ldots;$$

- But:

$$\text{int int} \quad t \quad = \quad k[8];$$

will

- Throw
    - An:
        - Exception.
- And
    - We
        - Say
            - That:

$$[i\ ..\ j]$$

is

- Equivalent
    - To:

$$[i],\qquad [i+1],\qquad \ldots,\qquad [j-1].$$

- And:

$$[i\ ..\ ]$$

- To:

$$[i],\qquad [i+1],\qquad \ldots.$$

- And:

$$[\ ..\ j]$$

- To:

$$\ldots,\qquad [j-2],\qquad [j-1].$$

- And:

$$[\,]$$

- To:

$$\ldots, \qquad [-2], \qquad [-1], \qquad [0], \qquad [1], \qquad [2], \qquad \ldots.$$

- And
    - So
        - We see that:

$$\mathrm{k[i\ ..\ j]}$$

is

- Equivalent
    - To:

$$\mathrm{k[i]}, \qquad \mathrm{k[i\ +\ 1]}, \qquad \ldots, \qquad \mathrm{k[j\ -\ 1]},$$

- And:

$$\mathrm{k[i\ ..\ ]}$$

- To:

$$\mathrm{k[i]}, \qquad \mathrm{k[i\ +\ 1]}, \qquad \ldots, \qquad \mathrm{k[k.length\ -\ 1]},$$

- And:

$$\mathrm{k[\ ..\ j]}$$

- To:

$$\mathrm{k[0]}, \qquad \mathrm{k[1]}, \qquad \ldots, \qquad \mathrm{k[j\ -\ 1]},$$

- And:

$$\mathrm{k[\,]}$$

- To:

$$\mathrm{k}[0], \qquad \mathrm{k}[1], \qquad \ldots, \qquad \mathrm{k}[\mathrm{k.length} \;-\; 1],$$

- And:

$$\mathrm{k}[\,][1]$$

- To:

$$\mathrm{k}[0][1], \qquad \mathrm{k}[1][1], \qquad \ldots, \qquad \mathrm{k}[\mathrm{k.length} \;-\; 1][1],$$

- And:

$$\mathrm{k}[\,][1, \;\; 0]$$

- To:

$$\mathrm{k}[0][1, \;\; 0], \qquad \mathrm{k}[1][1, \;\; 0], \qquad \ldots, \qquad \mathrm{k}[\mathrm{k.length} \;-\; 1][1, \;\; 0],$$

- Since:

$$\ldots, \qquad \mathrm{k}[-2], \qquad \mathrm{k}[-1]$$

are

- All
  - Equivalent
    - ▹ To:

$$\mathrm{k}[0],$$

- And:

$$\mathrm{k}[\mathrm{k.length}], \qquad \mathrm{k}[\mathrm{k.length} \;+\; 1], \qquad \ldots$$

- To:

$$\mathrm{k}[\mathrm{k.length} \;-\; 1].$$

- And
  - So

▷ We see that:

```
[ ],          [ .. 8],        [8 .. ],          [i .. i + 1],

[ ][0],       [ .. 8][0],     [8 .. ][1],       [i .. i + 1][1, 0]
```

are

- Range
  - Operators,
    - ▷ And:

```
[0],          [0][0],         [i],          [i][0],          [1][3, 1]
```

are

- Location
  - Operator.
    - ▷ And:
      - So

we

- Can
  - Write:

```
int i = k[0][0];

k[10] = k[20][1, 0];
```

- And
  - Also
    - ▷ We see that:

```
[i .. j]
```

is

- Something
  - That
    - ▹ Wraps:

$$[i], \quad [i + 1], \quad \ldots, \quad [j - 1].$$

- And:

$$k$$

is

- Something
  - That
    - ▹ Wraps:

$$k[0], \quad k[1], \quad \ldots, \quad k[k.length - 1].$$

- And
  - So
    - ▹ We see that:

$$k, \quad k1 \quad \textit{and} \quad k2$$

are

- Wrappers,
  - But:

$$k[\,] \quad \textit{and} \quad k[10 \;..\;]$$

are

- Enumerations.
  - And

▹ So:
  – If

we

- Write:

$$k[\,] \qquad or \qquad k[i \;..\; j],$$

it

- Will
  - Be
    ▹ Like:
      – Removing

the

- Wrapper
  - Called:

$$k,$$

- And
  - Enumerating
    ▹ All:
      – Elements

in

- That
  - Range
    ▹ At:
      – That

very

- Place
    - It
        - ▹ Is:
            - Written.
- And
    - So
        - ▹ If:

$$k1 \quad = \quad k2[\,]; \qquad (15)$$

the

- Elements
    - Of:

$$k2$$

will

- Be
    - Enumerated:
        - ▹ After

the

- Assignment
    - Operator.
        - ▹ And:
            - So

the

- Elements
    - Of:

k2

will

- Become
  - A list
    - ▹ Literal
      - – For:

k1,

which

- Inturn
  - Would
    - ▹ Be:
      - – Equivalent

to

- Saying
  - That,
    - ▹ A copy
      - – Of:

k2

is

- Made,
  - And
    - ▹ Given
      - – To:

k1.

- And so
    - If
        - ▷ We
            - Execute statement 15:

k1

will

- Point
    - To
        - ▷ A copy
            - Of:

k2.

- And
    - So
        - ▷ In:
            - General,

a

- Copy
    - Of
        - ▷ All:
            - Lists

to

- Which
    - A range
        - ▷ Operator:
            - Has

been

- Juxtaposed
    - Will
        - ▷ Be:
            - Made.
- And
    - If:

$$j >= i,$$

the

- Lengths:
    - Of

$$[j \ .. \ i] \qquad \textit{and} \qquad k[j \ .. \ i]$$

will

- Be:

$$0.$$

- And
    - So
        - ▷ If:

$$k = k1[j \ .. \ i];$$

- Then:

$$k.length == 0.$$

- And
    - If:

k.length == 0 *and* k1.length == 10,

- Then:

k, k[ ], k[10 .. 20], k1[10 ..],

will

- Be
    - Considered
        - As:
            - Empty-sets,
- And
    - Will
        - Not:
            - Throw

any

- Exception.
    - And
        - We
            - Can write.

```
k = k[ .. 2], 80 80, k[2 .. ];   // Insert after: k[1].
k = k[ .. 2], k[3 .. ];          // Delete: k[2].
```

- Let:

k5 *and* k6

be

- Instances
  - Of:

[int].

- Then
  - Since
    - ▹ In:

k5 += k6;

a

- Copy
  - Of:

k6

will

- Be
  - Appended
    - ▹ To:

k5,

we

- Can
  - Say
    - ▹ That:

k5 = k6 + i;

is

- Equivalent

- To.

$$k5 \quad = \quad k6, \quad i;$$

- But
  - We
    - Avoid:
      - It.
- And
  - So
    - We:
      - Do

not

- Allow
  - Statements
    - Like.

$$k5 \quad = \quad k6[\,] \quad + \quad i;$$

$$k5 \quad = \quad k6 \quad + \quad k6[\,];$$

- But
  - We
    - Can
      - Write.

$$k5 \quad = \quad k6, \quad i \; + \; j, \quad k6;$$

- We
  - Had
    - Stated:

        - That,

when

- We
    - Write:

$$k[\,][0],$$

the

- Elements
    - Of:

$$k[\,][0]$$

will

- Be
    - Enumerated.
        - ▹ And:
            - So

if

- We
    - Write:

$$k[\,][0] \quad = \quad \ldots;$$

we

- Will
    - Be:
        - ▹ Enumerating

all

- Locations
    - In:

$$k[\,][0],$$

on

- The
    - Left
        - ▹ Hand:
            - Side,
- And
    - The
        - ▹ Expression:
            - On

the

- Right
    - Hand
        - ▹ Side:
            - Will

be

- Applied
    - To
        - ▹ All:
            - Those locations.
- And so
    - The
        - ▹ Interpretation

          - Of:

$$k[\,][0] \quad = \quad i;$$

- Is:

  “*replace all elements of*: k[ ][0] *with*: i,”

- And
  - That
    - Of:

$$k[\,][0] \quad += \quad i \quad : \quad (k[\,][0] \quad < \quad 20);$$

- Is:

  “*add*: i *to all locations in*: k[ ][0]

  that satisfy: k[ ][0] < 20.”

- And
  - We
    - Can
      - Write:

```
k[ ][0]    =    i    :    (bool-Exp-Involving-k[ ][1]);

k[ ][0]++;

k[ ][0]++            :    (...);
```

- And
  - In:

$$k1[\,] \quad = \quad k2[\,];$$

we

- Can
    - Say
        - ▹ That:
            - If

the

- Length
    - Of
        - ▹ The:
            - Ranges

on

- Both sides
    - Are
        - ▹ The same,
            - Then:

$$\text{“}\textit{the} \quad i^{th} \quad \textit{element}\text{”}$$

on

- The
    - Right
        - ▹ Hand:
            - Side

will

- Replace
    - The
        - ▹ Corresponding:

      - Element

on

- The
    - Left
        - Hand:
            - Side.
- But
    - Since
        - It:
            - Will complicate,

we

- Say
    - That,
        - The:
            - Range-operator

can

- Only
    - Be
        - Used:
            - On

one

- Side
    - Of
        - The:
            - Assignment-operator.

- And
  - So
    - ▹ We:
      - Do

not

- Allow
  - Statements
    - ▹ Like:

$$\begin{aligned} \mathrm{k1}[\,] &= \mathrm{k2}[\,]; \\ \mathrm{k1}[\,][0] &= \mathrm{k2}[\,][0] + 10; \end{aligned}$$

- But
  - We
    - ▹ Can
      - Write:

$$\begin{aligned} \mathrm{k1} &= \mathrm{k2}[\,]; \\ \mathrm{k1}[\,][0] &= \mathrm{i}; \\ \mathrm{k1}[0] &= \mathrm{k2}[0]; \\ \mathrm{k1}[0][0] &= \mathrm{k2}[0][1]; \end{aligned}$$

- Since:

$$[0], \quad [0][0] \quad \textit{and} \quad [0][1]$$

are

- Location
  - Operators.
    - ▷ And:
      - To

avoid

- Complications,
  - We
    - ▷ Do:
      - Not

allow

- Statements
  - Like:

$$k[10\ ..\ ] \quad = \quad k1;$$

- And
  - Similarly,
    - ▷ We:
      - Do

not

- Allow
  - Statements
    - ▷ Like:

$$\begin{array}{lcr} k1[\,] & += & k2[\,]; \\ k1[\,][0] & += & k2[\,][0] \ + \ 10; \\ k[10\ ..\ ] & += & k1; \end{array}$$

- But
  - We
    - ▹ Can
      - Write.

$$k1 \quad += \quad k2[\,];$$

$$k1[\,][0] \quad += \quad i;$$

$$k1[0] \quad += \quad k2[0];$$

$$k1[0][0] \quad += \quad k2[0][1];$$

- The
  - Interpretation
    - ▹ Of:

$$k[\,] \quad \% = \quad 10 \;\; 10, \quad 20 \;\; 20;$$

- Is:

“*delete*: 10 10 *and* 20 20 *from*: k,”

- And
  - That
    - ▹ Of:

$$k[\,] \quad \% = \quad k[\,] \quad : \quad (k[\,][0] \; > \; 10);$$

- Is:

“*delete from*: k[ ] *where*: k[ ][0] > 10.”

- And

- That
  - Of:

$$k = k1[\,] : (k1[\,][0] == 20 \;||\; k1[\,][1] == 30);$$

- Is:

“*select* $*$ *from*: k1 *where*: k1[ ][0] == 20 *or* k1[ ][1] == 30.”

- Other
  - Examples
    - Are.

$$k = k1[\,][1,\; 0] : (k1[\,][0] == 20 \;||\; k1[\,][1] == 30);$$

$$k = k1[\,] : (k1[\,][0] \text{ in } (k2[\,][1] : (k2[\,][0] > 30)));$$

$$i = (k[\,] : (k[\,][1] < 10)).\text{max};$$

$$i = (k[\,] : (k[\,][1] < 10)).\text{min};$$

$$i = (k[\,] : (k[\,][1] < 10)).\text{sum};$$

$$i = (k[\,] : (k[\,][1] < 10)).\text{length};$$

$$i = \text{k.max} + \text{k.min} + \text{k.sum};$$

- Let:

k4

be

- An
  - Instance

▷ Of:

$$[\text{int} \quad \text{int} \quad \text{int} \quad \text{int}].$$

- Then:

$$\text{k4} \quad = \quad \text{k1} \ * \ \text{k2};$$

can

- Be
  - Used
    - ▷ To:
      - Generate

the

- Cross
  - Product
    - ▷ Of:

$$\text{k1} \qquad \textit{and} \qquad \text{k2}.$$

- Other
  - Examples
    - ▷ Are:

$$\text{k} \quad = \quad (\text{k1}[\,] \ * \ \text{k2}[\,])[\,][3, \ 0] \ : \ (\text{k1}[\,][1] \ == \ \text{k2}[\,][0]);$$

$$\text{k4} \quad = \quad \text{k1} \ * \ \text{k2}[\,], \quad \text{k1}[\,] \ * \ \text{k2}[\,], \quad 10 \ \ 10 \ \ 10 \ \ 10;$$

- And
  - Integer
    - ▷ Multiplication:

- Will

be

- Performed
    - In:

$$\text{int} \quad i \quad = \quad \text{k1}[0][0] \ * \ \text{k2}[0][0];$$

- And:

$$[\text{int} \ \ \text{int}] \quad k \quad = \quad \text{k1}[0] \ * \ \text{k2}[0];$$

is

- Equivalent
    - To:

$$\text{int} \ \ \text{int} \quad t \quad = \quad \text{k1}[0] \ * \ \text{k2}[0];$$

$$[\text{int} \ \ \text{int}] \quad k \quad = \quad t;$$

- And
    - If:

$$[\text{int} \ \ \text{int}] \quad k \quad = \quad \text{k1}[0 \ .. \ 1][0] \ * \ \text{k2}[0 \ .. \ 1][1];$$

then

- Cross
    - Product
        - ▹ Operation:
            - Will

be

- Performed.
    - But:

$$[\text{int} \;\; \text{int}] \quad k \quad = \quad k1[0][0] \; * \; k2[0][0];$$

will

- Not
    - Compile,
        - Since:

$$\text{int} \qquad i \quad = \quad k1[0][0] \; * \; k2[0][0];$$

$$[\text{int} \;\; \text{int}] \quad k \quad = \quad i;$$

will

- Not
    - Compile.
        - The
            - Interpretation of:

$$k \quad = \quad k1[\,] \quad : \quad (\ldots), \quad (<)k1[\,][0];$$

- Is:

"*select* $*$ *from*: k1 ... *order by*: k1[ ][0] *asc*."

- Let:

$$k3$$

be

- An

- Instance
  - Of:

$$[\text{int} \quad \text{int} \quad \text{int}].$$

- Then
  - We
    - Allow
      - Statements like:

$$\begin{array}{lllllr}
[\text{int} \quad \text{int}] & \text{k} & = & (\text{k1}[\,] * \text{k2}[\,])[\,][0, 3] & : & (\ldots), \quad (<)\text{k1}[\,][0], (>)\text{k2}[\,][3]; \\
[\text{int}] & \text{k5} & = & \text{k3}[\,][0], \quad 10 & : & (<)\text{k3}[\,][1]; \\
[\text{int} \quad \text{int}] & \text{k2} & = & \text{k3}[\,][1, \quad 0] & : & (<)\text{k3}[\,][0], \quad (>)\text{k3}[\,][1]; \\
\text{k3} & & = & \text{k3}[\,] & : & (<)\text{k3}[\,][2], \quad (>)\text{k3}[\,][0]; \\
\text{k2} & & = & \text{k3}[\,][1, \quad 0] & : & (\ldots), \quad (<)\text{k3}[\,][2];
\end{array}$$

- But
  - Not:

$$[\text{int} \quad \text{int}] \quad \text{k2} \quad = \quad \text{k3}[\,][1, \quad 0] \quad : \quad (<)\text{k3}[\,][0], \quad (\ldots);$$

- Let:

$$\text{k9}, \qquad \text{k10} \qquad \textit{and} \qquad \text{k11}$$

be

- Instances
  - Of:

$$[(\text{null})\text{int} \quad (\text{null})\text{int}].$$

- Then
  - We
    - ▹ Can
      - Write:

$$k9 = (k10[\,] * k11[\,])[\,][3, 0] : ((null)k10[\,][3] == k11[\,][0]);$$

for

- Left
  - Join.
    - ▹ And:
      - Similarly,

for

- The
  - Other:
    - ▹ Two
      - Joins.
- But:

$$k = (k1[\,] * k2[\,])[\,][3, 0] : ((null)k1[\,][3] == k2[\,][0]);$$

will

- Not
  - Compile.
    - ▹ Let:

$$\text{int intReturner(int);} \tag{16}$$

be

- Some

- ◦ Method.
  - ▹ Then:
    - – If

we

- Write:

$$\text{intReturner}(k[\,][0])$$

it

- Would
  - ◦ Mean:
    - ▹ That,

the

- Wrapper
  - ◦ Called:

$$k$$

is

- Removed,
  - ◦ And
    - ▹ We:
      - – Are

asking

- Method 16
  - ◦ To
    - ▹ Act:
      - – Individually

on

- All
    - Elements
        - In
            - The enumeration:

$$\text{k[ ][0].}$$

- And so
    - When
        - We
            - Execute:

$$\text{intReturner(k[ ][0]),}$$

we see that,

- Method 16
    - Will
        - Act:
            - On

all

- Elements
    - In
        - The
            - Enumeration:

$$\text{k[ ][0].}$$

- And
    - So
        - The:

- Result

will

- Be
  - Another:
    - Enumeration.
- And
  - So
    - If:

$$\text{k5} = \text{intReturner(k[ ][0]);}$$

then

- It
  - Will
    - Be:
      - Equivalent

to

- Saying
  - That:

$$\text{“}\textit{the} \quad i^{th} \quad \textit{element}\text{”}$$

- Of:

$$\text{k[ ][0]}$$

is

- Given
  - To:

        - ▹ Method 16,

- And
    - ◦ The
        - ▹ Result:
            - Of

that

- Operation
    - ◦ Is:

“*the* i^{th} *element*”

- Of:

k5.

- And so
    - ◦ When
        - ▹ We
            - Write:

k5 = intReturner(k[ ][0]);

we

- Say:

“k[ ][0] *is transformed to*: k5 *through*: intReturner.”

- And:

k5 = intReturner(k[0][0]), intReturner(k[ ][0]), intReturner(k[0][0]);

is

- Like.

```
        k5   =   <some-int>,   <some-list>,   <some-int>;
```

- Let:

```
            int              intReturner(int,  int);

            int    intReturnerTwo(int,  int,  [int  int]);
```

be

- Methods.
    - The
        - Interpretation
            - Of:

```
                k5   =   intReturner(k[ ][0],  10);
```

- Is:

```
for(int  i  in  0  ..  k.length)

    k5  +=  intReturner(k[i][0],  10);
```

- And
    - That
        - Of:

```
        k5   =   intReturnerTwo(k[ ][0],  k[ ][1],  k1);
```

- Is:

```
for(int  i  in  0  ..  k.length)

    for(int  j  in  0  ..  k.length)

        k5  +=  intReturnerTwo(k[i][0],  k[j][1],  k1);
```

- And
    - That
        - Of:

```
k5 = intReturnerTwo(k[ ][0], k[1 .. ][1], k1);
```

- Is.

```
for(int i in 0 .. k.length)
    for(int j in 1 .. k.length)
        k5 += intReturnerTwo(k[i][0], k[j][1], k1);
```

- Note that,
    - When
        - There
            - Are:

“m *lists*”

- And:

$$n_i$$

different

- Ranges
    - Are
        - Used
            - With:

“*the* i^{th} *list*,”

there

- Will
  - Be:

$$n_1 \quad + \quad n_2 \quad + \quad \dots \quad + \quad n_m$$

number

- Of:
  - For
    - ▷ Loops.
- And
  - If:

$$\text{k.length} \ == \ 0,$$

- And
  - We
    - ▷ Execute:

$$\text{k5} \quad = \quad \text{intReturner(k[ ][0]);}$$

- Then:

$$\text{k5.length} \ == \ 0,$$

- And
  - There
    - ▷ Will:
      - Be

no

- Exception.
  - And

        ▷ We:
            – Can

say

- That,
    ◦ The
        ▷ Interpretation
            – Of:

$$[\text{int}] \quad \text{k5} \quad = \quad \text{k}[\,][0] \; + \; 8; \tag{17}$$

- Is:

$$\text{“}\textit{for all} \quad \text{i}, \quad \text{k5}[\text{i}] \; == \; \text{k}[\text{i}][0] \; + \; 8.\text{”}$$

- But
    ◦ We
        ▷ Avoid:
            – Statements

like

- Statement 17.
    ◦ Or
        ▷ To:
            – Implement statement 17,

we

- First
    ◦ Write:

$$\text{int} \;\; \text{intReturner(int} \;\; \text{i}, \;\; \text{int} \;\; \text{j})\{ \;\; \text{return} \;\; \text{i} \; + \; \text{j}; \;\; \};$$

- And

- ◦ Then.

$$[\text{int}] \quad k5 \quad = \quad \text{intReturner}(k[\,][0], \quad 8);$$

- Let:

$$\text{int} \quad \text{intReturnerThree}([\text{int}]); \tag{18}$$
$$[\text{int}] \quad \text{listReturner}(\text{int});$$

be

- Methods.
  - ◦ Then
    - ▹ If:

$$\text{intReturnerThree}(k5[\,]) \tag{19}$$

we see that,

- We
  - ◦ Are
    - ▹ Asking:
      - Method 18

to

- Act
  - ◦ On
    - ▹ An:
      - Enumeration,
- And
  - ◦ Not
    - ▹ On:
      - A list.
- And

- So
  - ▹ There:
    - Will

be

- An
  - Error.
    - ▹ And:
      - So expression 19

should

- Be
  - Rewritten
    - ▹ As:

$$\text{intReturnerThree}(\text{k5}).$$

- And
  - If:

$$\text{listReturner}(\text{k5}[\,])$$

a

- List
  - Will
    - ▹ Be:
      - Generated

for

- Each
  - Element

        - ▹ In:

k5.

- And
    - So
        - ▹ The:
            - Result

will

- Be
    - An
        - ▹ Instance
            - Of:

[[int]].

- And
    - So
        - ▹ Using:
            - These,

we

- Can
    - Distinguish:
        - ▹ Between

the

- Invocations
    - Of:

int someMethod(int);

int someMethod([int]);

[int] someMethod(int);

[int] someMethod([int]);

- And
  - In:

$$k = k1[\,] : (\ldots);$$

the

- Things
  - Written
    - ▹ In
      - Between:

‘:’ *and* ‘;’

is

- Called
  - The:
    - ▹ Condition
      - Part.
- Let:

int f(int, int, [int int])

be

- Some
  - Method,

- ▹ And
  - Let:

$$S \quad = \quad \{ \quad \text{k1[0]} \times \text{k2[1]} \quad | \quad \text{k1[ ][1]} \;==\; \text{k2[ ][0]} \quad \},$$

- And
  - Assume:

$$S \quad = \quad \{ \quad e_0[0] \;\; e_0[1], \quad e_1[0] \;\; e_1[1], \quad \ldots \quad \},$$

- And
  - Let:

$$(e_0[0],\;\; e_0[1],\;\; \text{k}) \quad \xrightarrow{\text{f}} \quad e'_0$$

$$(e_1[0],\;\; e_1[1],\;\; \text{k}) \quad \xrightarrow{\text{f}} \quad e'_1$$

$$\vdots$$

- And
  - We
    - ▹ Execute:

[int] result = f(k1[ ], k2[ ], k) : (k1[ ][1] == k2[ ][0]);

- Then:

$$\text{result} \quad = \quad \{ \quad e'_0, \quad e'_1, \quad \ldots \quad \}.$$

- And
  - So
    - ▹ We see that:
      - If

a

- Range
    - Operator
        - ▷ Is:
            - Juxtaposed

to

- The
    - Right:
        - ▷ Of

a

- List,
    - The
        - ▷ Condition:
            - In

the

- Condition
    - Part
        - ▷ Will:
            - Be applied

to

- That list,
    - And
        - ▷ The:
            - Respective elements

will

- Be

- ◦ Chosen.
  - ▹ And:
    - – So

we

- Write:

$$k = (k1 * k2[\,])[0\ \ 3],\quad 10\ \ 10 \ : \ (k2[\,][0] == 20);$$

$$k = (k1[\,] * k2[\,])[\,][0,\ 3] \ : \ (k1[\,][1] == k2[\,][0]);$$

- And
  - ◦ Not.

$$k = (k1 * k2)[\,][0,\ 3] \ : \ (k1[\,][1] == k2[\,][0]);$$

- And
  - ◦ In:

$$k = k1[\,] \ : \ (k2[\,][0] == 10);$$

- Since:

k1 *and* *the condition-part*

are

- Independent
  - ◦ Of
    - ▹ Each:
      - – Other,

the

- Condition

- In
  - The:
    - Condition-part

will

- Not
  - Be
    - Applicable
      - To:

k1.

- And so
  - To
    - Avoid:
      - Errors,

we

- Say
  - That,
    - The:
      - Condition

in

- The
  - Condition
    - Part:
      - Should

be

- Applicable

- To
  - At:
    - Least

one

- List.
  - And so
    - In
      - Statements like:

$$k \quad = \quad k1[8 ..] \quad : \quad (\ldots); \tag{20}$$

the

- Range
  - Associated
    - With:

$$k1$$

should

- Be
  - Exactly
    - The:
      - Same

as

- That
  - In
    - The:
      - Condition-part.
- But

- We
    - Can
        - Write:

$$k = k1[8\ ..\ ] : (k1[8\ ..\ ][0] \text{ in } k1[\ ..\ 20][0]);$$

- And:

$$[\text{int}]\ k5 = k3 : (<)k3[\,][1];$$

should

- Be
    - Rewritten
        - As:

$$[\text{int}]\ k5 = k3[\,][1] : (<)k3[\,][1];$$

- And
    - If:

$$k3 = 10\ 100\ 1000,\ 20\ 200\ 2000,\ 30\ 300\ 3000;$$

- Then:

$$10\ 1000 \text{ in } k3[1\ ..\ ][0,\ 2] \qquad 999\ 9999 \text{ in } k3[1\ ..\ ][1,\ 2]$$

$$999\ 9999\ !\text{in } k3[1\ ..\ ][1,\ 2] \qquad 10\ 100\ 1000 \text{ in } k3$$

will

- Be:

false, false,

true, *and* true

respectively.

- And:

$$(\mathrm{k5} \quad == \quad \mathrm{k6}) \quad == \quad \mathrm{true},$$

- If:

$$\mathrm{k5} \qquad \textit{and} \qquad \mathrm{k6}$$

have

- The
  - Same
    - ▷ Elements:
      - Arranged

in

- The
  - Same:
    - ▷ Order.
- And
  - So
    - ▷ If:

$$[\mathrm{int}] \quad \mathrm{k5} \quad = \quad 10, \quad 20, \quad \mathrm{k6} \quad = \quad 20, \quad 10;$$

- Then:

$$\mathrm{k5} \quad == \quad \mathrm{k6} \qquad \textit{and} \qquad \mathrm{k5} \quad != \quad \mathrm{k6} \tag{21}$$

will

- Be:

$$\mathrm{false} \qquad \textit{and} \qquad \mathrm{true}$$

respectively.

- And:

$$<, \qquad >, \qquad <=, \qquad >=$$

can

- Be
  - Used
    - For:
      - Proper sublist operation,
      - Proper super list operation,
      - Is equal to or a proper sub list operation,
      - Is equal to or a proper super list operation,
- And:

$$===, \qquad =!=, \qquad <<<, \qquad >>>, \qquad =<=, \qquad =>=$$

for

- Set equivalence,
- Set non-equivalence,
- Proper subset operation,
- Proper superset operation,
- Is equal to or a proper subset operation,
- Is equal to or a proper superset operation.
  - Respectively.
    - And so
      - If:

$$[\text{int}] \quad k5 \quad = \quad 10, \quad 20, \quad k6 \quad = \quad 20, \quad 10;$$

- Then:

k5 === k6, k5 =<= k6, k5 <<< k6 (22)

will

- Be:

true, true, false

respectively.

- And
  - In
    - ▷ Expressions 21 and 22:

==

will

- Be
  - Used
    - ▷ To:
      - Check

for

- Equivalence
  - Between
    - ▷ The
      - Elements in:

k5 *and* k6.

- And:

&!&, &&&, &=&

for

- Does not intersect,
- Proper set intersection,
- Intersects or are equal as sets,
  - Respectively.
    - And:

$$(\text{<list}_1\text{>} \quad \text{<boolean-set-operator>} \quad \text{<list}_2\text{>})\text{.multiset}$$

for

- Multiset
  - Operations.
    - Exemplifying,
      - If:

$$[\text{int}] \quad \text{k5} \quad = \quad 10, \quad 20, \quad 10, \quad \text{k6} \quad = \quad 20, \quad 10;$$

- Then:

$$\text{k5} \ \ \&\&\& \ \ \text{k6} \qquad \textit{and} \qquad (\text{k5} \ \ \&\&\& \ \ \text{k6})\text{.multiset}$$

will

- Be:

$$\text{false} \qquad \textit{and} \qquad \text{true,}$$

- Since:

$$\text{k5} \ \ \&{=}\& \ \ \text{k6} \qquad \textit{and} \qquad (\text{k5} \ \ \&{=}\& \ \ \text{k6})\text{.multiset}$$

- Are:

$$\text{true} \qquad \textit{and} \qquad \text{false}$$

respectively.

- Or
    - Multiset
        - ▷ Operation:
            - Is

like,

- Attaching
    - An
        - ▷ Unique:
            - Identifier

to

- All
    - Elements
        - ▷ Before:
            - Performing

the

- Operation.
    - And:

$$\text{k5} == \text{k6}[\,] \qquad \textit{and} \qquad \text{k5}[\,] == \text{k6}[\,]$$

are

- Equivalent
    - To:

$$\text{k5} == \text{k6},$$

- And
    - Similarly

            ▹ For:
                – Others.

- And
    - Since
        ▹ In:

$$k[0]$$

we

- Locate
    - An
        ▹ Element,
            – We see that:

$$[k]$$

is

- Like
    - Locating
        ▹ The address
            – Of:

$$k.$$

- And
    - So:

$$([k1] == [k2]) == true,$$

- If:

$$k1 \quad \textit{and} \quad k2$$

points

- To
  - The
    - Same:
      - Location.
- And:

$$\mathrm{k1} \quad = \quad [\mathrm{k2}]; \qquad \textit{and} \qquad \mathrm{k1} \quad = \quad \mathrm{k2};$$

are

- Equivalent.
  - And:

$$\mathrm{k} \quad = \quad \mathrm{k1}, \quad [\mathrm{k2}];$$

is

- Equivalent
  - To:

$$\mathrm{k} \quad = \quad \mathrm{k1}, \quad \mathrm{k2};$$

- And:

$$[\mathrm{k}][\,] \qquad \textit{and} \qquad [\mathrm{k}][\,][0]$$

- To:

$$\mathrm{k}[\,] \qquad \textit{and} \qquad \mathrm{k}[\,][0]$$

$$\textit{respectively}.$$

- And:

$$[\mathrm{int}] \quad \mathrm{k5} \quad = \quad 10, \ 20, \ 30;$$

$$\mathrm{if} \ \ (\mathrm{k5} \ \ == \ \ (10, \ 20, \ 30))\{\ldots\}$$

should

- Be
  - Rewritten
    - ▷ As.

$$[\text{int}] \quad \text{k5} \quad = \quad 10, \ 20, \ 30;$$

$$[\text{int}] \quad \text{k6} \quad = \quad 10, \ 20, \ 30;$$

$$\text{if} \quad (\text{k5} \ == \ \text{k6})\{\ldots\}$$

- Let:

$$\begin{aligned} \text{boolean} &\quad \text{boolReturner(int);} \\ \text{boolean} &\quad \text{boolReturnerTwo(int, int);} \\ \text{boolean} &\quad \text{boolReturnerThree(int int);} \\ \text{boolean} &\quad \text{boolReturnerFour(int int, int);} \\ \text{boolean} &\quad \text{boolReturnerFive(boolean);} \end{aligned}$$

be

- Methods,
  - And
    - ▷ Let:

$$b$$

be

- Of

  - Type:

boolean.

- The
  - Interpretation
    - ▷ Of:

```
b = boolReturner(k[ ][0]);
```

- Is:

```
b = true;

for(int i in 0 .. k.length)

    if (boolReturner(k[i][0]) == false)

        b = false;
```

- And
  - That
    - ▷ Of:

```
b = boolReturnerTwo(k[ ][0], k[ ][0]);
```

- Is:

```
b = true;

for(int i in 0 .. k.length)

    if (boolReturnerTwo(k[i][0], k[i][0]) == false)

        b = false;
```

- And
    - That
        - ▹ Of:

```
        b = boolReturnerTwo(k[ ][0], k[ ][1]);
```

- Is:

```
b = true;
for(int i in 0 .. k.length)
    for(int j in 0 .. k.length)
        if (boolReturnerTwo(k[i][0], k[j][1]) == false)
            b = false;
```

- And
    - That
        - ▹ Of:

```
        b = boolReturnerTwo(k[ ][0], k[1 .. ][0]);
```

- Is:

```
b = true;
for(int i in 0 .. k.length)
    for(int j in 1 .. k.length)
        if (boolReturnerTwo(k[i][0], k[j][0]) == false)
            b = false;
```

- And
  - That
    - ▷ Of:

```
b = boolReturnerTwo(k1[ ][0], k2[ ][0]);
```

- Is:

```
b = true;
for(int i in 0 .. k1.length)
    for(int j in 0 .. k2.length)
        if (boolReturnerTwo(k1[i][0], k2[j][0]) == false)
            b = false;
```

- And
  - That
    - ▷ Of:

```
b = boolReturnerThree(k[ ]);
```

- Is:

```
b = true;
for(int i in 0 .. k.length)
    if (boolReturnerThree(k[i]) == false)
        b = false;
```

- And
    - That
        - ▷ Of:

```
b = boolReturnerFour(k[ ], k[ ][0]);
```

- Is:

```
b = true;
for(int i in 0 .. k.length)
    for(int j in 0 .. k.length)
        if (boolReturnerFour(k[i], k[j][0]) == false)
            b = false;
```

- And
    - As
        - ▷ Before:
            - When

there

- Are:

“m *lists,*”

- And:

$$n_i$$

different

- Ranges

- Are
  - Used
    - With:

“*the* i^{th} *list*,”

there

- Will
  - Be:

$$n_1 \quad + \quad n_2 \quad + \quad \ldots \quad + \quad n_m$$

number

- Of
  - For
    - Loops.
      - Let:

bl1 *and* bl2

be

- Instances
  - Of:

[boolean].

- The
  - Interpretation
    - Of:

b = bl1[ ];

- Is:

```
b          =    true;

for(int  i  in  0  ..  bl1.length)

    if  (bl1[i]  ==  false)

        b    =    false;
```

- And
    - That
        - ▹ Of:

```
                b    =    bl1[ ]  &&  bl2[ ];
```

- Is:

```
boolean    b1    =    bl1[ ];

boolean    b2    =    bl2[ ];

b                =    b1  &&  b2;
```

- And
    - That
        - ▹ Of:

```
  b    =    boolReturner(k[ ][0])  &&  boolReturner(k[ ][1]);
```

- Is:

```
boolean    b1    =    boolReturner(k[ ][0]);

boolean    b2    =    boolReturner(k[ ][1]);

b                =    b1  &&  b2;
```

- And
    - Similarly,
        - For:
            - Others.
- And
    - We
        - Do not
            - Allow:

$$\text{b} \quad = \quad \text{bl1;}$$

since

- We
    - Do
        - Not
            - Allow:

$$\text{b} \quad = \quad \text{boolReturnerFive}(\text{bl1});$$

- And
    - If:

$$\text{b} \quad = \quad \text{boolReturner}(\text{k}[\,][0]) \quad (8) == \quad \text{true;}$$

- Then:

$$\text{b} \; == \; \text{true,}$$

if

- At
    - Least
        - Eight

      - Elements of:

$$k[\,][0]$$

- Satisfy.

$$\text{boolean}\quad \text{boolReturner(int);}$$

- And
  - So
    - ▷ If:

$$\text{k.length} \quad == \quad 100,$$

- And
  - Only
    - ▷ Eighty
      - Elements of:

$$k[\,][0]$$

- Satisfy:

$$\text{boolean}\quad \text{boolReturner(int);}$$

- Then:

$$(\text{boolReturner}(k[\,][0]) \quad (800) == \quad \text{true}) \quad == \quad \text{false}.$$

- The
  - Interpretation
    - ▷ Of:

$$b \quad = \quad \text{intReturner}(k[\,][0]) \quad (8) == \quad 10;$$

- Is:

```
[int]   k5   =   intReturner(k[ ][0]);

b            =   k5[ ]  (8)==  10;
```

- And:

$$(k[\,][0] \quad ([8]) == \quad 10) \quad == \quad \text{true},$$

if

- Exactly
  - Eight
    - ▹ Elements
      - Of:

$$k[\,][0]$$

are

- Equal
  - To:

$$10,$$

- And:

$$(k[\,][0] \quad ([8 \quad .. \quad 10]) == \quad 10)$$

is

- Equivalent
  - To:

$$(k[\,][0] \quad ([8]) == \quad 10) \quad || \quad (k[\,][0] \quad ([9]) == \quad 10).$$

- And
  - If:

$$\text{k.length} \;==\; 0,$$

- Then:

$$(\text{boolReturner}(\text{k}[\,][0])\;\;(<\text{some-int}>)==\;\;<\text{some-value}>)\;\;==\;\;\text{true}.$$

- And:

$$\text{i} \;=\; -1;$$
$$\text{b} \;=\; \text{boolReturner}(\text{k}[\,][0])\;\;(\text{i})==\;\;\text{true};$$

is

- Equivalent
  - To:

$$\text{b} \;=\; \text{boolReturner}(\text{k}[\,][0])\;\;(0) ==\;\;\text{true}; \tag{23}$$

- And
  - Statement 23
    - ▹ To:

$$\text{b} \;=\; \text{true};$$

- And
  - If:

$$([\text{j}\;\;..\;\;\text{i}]).\text{length}\;\;==\;\;0.$$

- Then:

$$(\text{k}[\,][0]\;\;([\text{j}\;\;..\;\;\text{i}])==\;\;10)\;\;==\;\;\text{true}.$$

- And
  - We

- ▹ Do not
  - Allow:

$$[\text{int int}] \quad k1 = \ldots, \quad k2 = \ldots;$$

$$b = k1 \;(8){<}\; k2;$$

- And
  - Similarly,
    - ▹ For:
      - Others.
- And
  - If:

$$k \;|{=}\; k1;$$

the

- Elements
  - Of:

$$k1$$

that

- Are
  - Not
    - ▹ Present
      - In:

$$k$$

will

- Be
  - Append
    - To:

$$k$$

without

- Repetition.
  - And:
    - So

the

- Old
  - Elements
    - In:

$$k$$

will

- Remain
  - As:
    - Such.
- And
  - So:

$$k \quad += \quad k1 \;\; || \;\; k2;$$

is

- Equivalent
  - To.

```
[int  int]  kTemp      =     k1[ ];

kTemp                  |=    k2;

k                      +=    kTemp;
```

- Other
  - Similar,
    - Operations
      - Are:

```
k     =     k1  &&  k2;          //  Set  intersection.

k     =     k1  %  k2;           //  Set  difference.

k     =     k  ||  k1  ||  k2;

k     =     k1[ ]  &&  k2,    k1[ ]  ||  k2[ ],    10  10;
```

- And:

```
k     |=     k1[ ],      k1[ ]  ||  k2;

k     =      k1[ ]    &&      20  20,       30  30;

k     =      k1[ ]    ||      20  20,       30  30;

k     =      k1[ ]    %       20  20,       30  30;

k5    =      i  %  j  %  k6  %  j  %  i;

k     =      k  %  k1  &&  k2  ||  k1  &&  k2;
```

is

- Equivalent
    - To:

$$\text{k} \quad |= \quad \text{k1}, \quad \text{k1} \ || \ \text{k2};$$

$$\text{k} \quad = \quad (\text{k1} \ \&\& \ 20 \ 20), \quad 30 \ 30;$$

$$\text{k} \quad = \quad (\text{k1} \ || \ 20 \ 20), \quad 30 \ 30;$$

$$\text{k} \quad = \quad (\text{k1} \ \% \ 20 \ 20), \quad 30 \ 30;$$

$$[\text{int}] \ \text{k7} \quad = \quad \text{i} \ \% \ \text{j}, \quad \text{k8} \ = \ \text{j} \ \% \ \text{i};$$

$$\text{k5} \quad = \quad (\text{k7} \ \% \ \text{k6}) \ \% \ \text{k8};$$

$$\text{k} \quad = \quad ((\text{k} \ \% \ \text{k1}) \ \&\& \ \text{k2}) \ || \ (\text{k1} \ \&\& \ \text{k2});$$

- And:

$$\text{k} \quad = \quad \text{k1}[\,] \ \&\& \ \text{k2}[\,], \quad \text{k1}[\,] \ || \ \text{k2}[\,], \quad \text{k1}[\,] \ \% \ \text{k2}[\,] \quad : \quad (\ldots);$$

is

- Like:

$$\text{k} \quad = \quad \text{k1}[\,][0] \ * \ \text{k2}[\,][1], \quad \text{k1}[\,][0] \ * \ \text{k2}[\,][1] \quad : \quad (\ldots);$$

- And
  - We
    - ▷ Do:
      - Not

allow

- Statements
  - Like:

$$\text{k}[\,] \quad \&= \quad \text{k1}[\,]; \qquad \textit{and} \qquad \text{k}[10 \ .. \ ] \quad \&= \quad \text{k1};$$

- And

- Similarly,
  - For:

$$|= \qquad \textit{and} \qquad \%=.$$

- But
  - We
    - Can
      - Write:

```
[boolean]  bl  =        ...;
bl[ ]          &=       bl[0]    :   (...);
bl[ ]          |=       bl[0]    :   (...);
k[ ][0]        %=       k[0][0]  :   (...);
```

- And
  - If:

```
[int]  k5  =  10, 20, 10, 10,   k6  =  10, 30, 10;
```

- Then:

(k5 && k6).multiset *and* (k5 % k6).multiset

will

- Be
  - Equal
    - To:

$$10,\ 10 \qquad \textit{and} \qquad 20,\ 10$$

respectively.

- And
  - If:

$$\text{k5} \quad = \quad \text{k[ ][0]} \quad == \quad \text{i} \;\; || \;\; \text{k[ ][0]} \quad == \quad \text{j}; \tag{24}$$

- Then:

$$\text{k5}$$

will

- Hold
  - All
    - ▷ Indices
      - Of:

$$\text{i} \qquad \textit{and} \qquad \text{j}$$

- In:

$$\text{k[ ][0]},$$

- And:

$$\text{k5.length} \quad == \quad 0,$$

- If:

$$\text{i} \;\; \text{!in} \;\; \text{k[ ][0]} \qquad \textit{and} \qquad \text{j} \;\; \text{!in} \;\; \text{k[ ][0]}.$$

- And
  - Similarly,
    - ▷ For.

$$\text{k5} \quad = \quad \text{k[ ][0]} \quad >= \quad \text{i} \;\; \&\& \;\; \text{k[ ][1]} \quad <= \quad \text{j}; \tag{25}$$

- The
    - Interpretation
        - ▹ Of:

$$\text{k5} = \text{boolReturner}(\text{k}[\,][0]); \tag{26}$$

- Is:

"*find the indices of all elements of*: k[ ][0] *that satisfy*: boolReturner."

- And
    - We
        - ▹ Say:
            - That,

in

- Statements
    - Like
        - ▹ In:
            - Statements 24, 25 and 26,

only

- One
    - Operand
        - ▹ Of:
            - Length

greater

- Than:

$$1$$

can

- Be
  - Used,
    - ▷ So that:
      - Index-operation

will

- Be
  - Done
    - ▷ In:
      - Exactly

one

- List.
  - But:

$$\text{k5} \quad = \quad \text{k1[ ][0]} \;==\; 8, \quad \text{k2[ ][0]} \;==\; 8;$$

is

- Equivalent
  - To:

$$\begin{aligned} \text{k5} \quad &= \quad \text{k1[ ][0]} \;==\; 8; \\ \text{k5} \quad &+= \quad \text{k2[ ][0]} \;==\; 8; \end{aligned}$$

- And
  - If:

$$\text{k5} \quad = \quad \text{k2} \;<\; \text{k1}; \tag{27}$$

- Then:

$$k5$$

will

- Hold
  - The
    - ▷ Indices:
      - Of

all

- Non
  - Intersecting
    - ▷ Occurrences
      - Of:

$$k2,$$

- And:

$$\text{k5.length} == 0,$$

- If:

$$\text{k2.length} == 0.$$

- And
  - We
    - ▷ Say:
      - That,

in

- Statements
  - Like
    - ▷ Statement 27:

– Only

one

- Operation
  - Using:

$$<$$

can

- Be
  - Written.
    - And:
      - So

we

- Do
  - Not
    - Allow
      - Statements like:

$$k5 = k2 > k1;$$

$$k5 = k2 \mathrel{!}= k1;$$

$$k5 = k2 == k1;$$

$$k5 = k2 >= k1;$$

$$k5 = k2 <= k1;$$

- But
  - We
    - Can

    - Write.

$$\text{k5} \quad = \quad \text{k2} \;<\; \text{k1}, \quad \text{k1[ ][0]} \;==\; \text{i};$$

- The
    - Interpretation
        - Of:

$$\text{k5} \quad = \quad \text{k2} \;\; (\&\&\&)< \;\; \text{k1};$$

- Is:

  "*find the indices of all occurrences of*: k2 *in*: k1."

- And
    - If:

$$\text{[int int]} \quad \text{k} \quad = \quad \text{k2} \;<\; \text{k1};$$

- Then:

$$\text{k[ ][0]}$$

will

- Hold:

  "*the lower-bounds*"

- And:

$$\text{k[ ][1]}$$

will

- Hold:

  "*the upper-bounds* + 1."

- The

- Interpretation
    - ▹ Of:

$$\mathrm{i} \quad = \quad \mathrm{k[\,][0]} \quad (8){==} \quad \mathrm{j;}$$

- Is:

“*find the*: $\quad 8 \quad + \quad 1^{th}$ *index of*: $\quad$ j $\quad$ *in*: $\quad$ k[ ][0],”

- And
  - That
    - ▹ Of:

$$\mathrm{k5} \quad = \quad \mathrm{k[\,][0]} \quad (8){==} \quad \mathrm{j;}$$

- Is:

“*find the first*: $\quad 8 \quad + \quad 1$ *indices of*: $\quad$ j $\quad$ *in*: $\quad$ k[ ][0].”

- And
  - Similarly,
    - ▹ For.

$$\mathrm{i} \quad = \quad \mathrm{k1} \quad (8){<} \quad \mathrm{k2;}$$

$$\mathrm{k5} \quad = \quad \mathrm{k1} \quad (8){<} \quad \mathrm{k2;}$$

- The
  - Interpretation
    - ▹ Of:

$$\mathrm{i} \quad = \quad \mathrm{k[\,][0]} \quad {==} \quad \mathrm{j;}$$

- Is:

“*find the last index of*: j *in*: k[ ][0],”

- And:

$$\text{k5} \quad = \quad 8 \;\; \text{in} \;\; \text{k[ ][0]}, \quad \text{k1} \;\; (-1)< \;\; \text{k2}, \quad \text{k2} \;\; (8)(\&\&\&)< \;\; \text{k1};$$

is

- Equivalent
  - To:

$$\text{k5} \quad = \quad \text{k[ ][0]} \;\; == \;\; 8, \quad \text{k1} \;\; (0)< \;\; \text{k2}, \quad \text{k2} \;\; (\&\&\&)(8)< \;\; \text{k1};$$

- And:

$$\text{i} \quad = \quad \text{caluse}_1, \quad \text{caluse}_2;$$

- To:

$$\text{i} \quad = \quad \text{caluse}_2;$$

- And:

$$\text{k5} \quad = \quad \text{k[ ][0]} \;\; (8)== \;\; 10 \;\; || \;\; \text{k[ ][0]} \;\; (9)== \;\; 100;$$

should

- Be
  - Rewritten
    - ▹ As.

$$\text{k5} \quad = \quad \text{k[ ][0]} \;\; (8)== \;\; 10 \;\; || \;\; \text{k[ ][0]} \;\; (8)== \;\; 100;$$

- The
  - Interpretation
    - ▹ Of:

$$\text{k(k1,} \;\; \text{k2);} \tag{28}$$

- Is:

“*replace all sublists of*: k *that matches*: k2 *by*: k1,”

- And
    - That
        - ▹ Of:

k(8, k1, k2);

are

- Similar,
    - Except
        - ▹ That:
            - Only

the

- First
    - Eight
        - ▹ Sublists:
            - Will

be

- Replaced.
    - And
        - ▹ In:

k(k1, k2, 8);

only

- The
    - Last

            ▹ Eight:
                – Sublists

will

- Be replaced.
    - And
        ▹ Similarly,
            – For:

$$k(8,\quad k1,\quad k2,\quad 9);$$

- And
    - In:

$$k([8],\quad k1,\quad k2);$$

only

- The
    - Eighth
        ▹ Sublist:
            – Will

be

- Replaced.
    - And
        ▹ Similarly,
            – For:

$$k([8],\quad k1,\quad k2,\quad [9]); \tag{29}$$

- Let:

k20

be

- An
    - Instance
        - Of:

$$[\text{int} \quad \text{int}].$$

- Then:

$$\text{int} \quad i \quad = \quad -1;$$

$$\text{k20} \qquad = \quad \text{k}(\text{i}, \ \text{k1}, \ \text{k2}, \ \text{i});$$

is

- Equivalent
    - To:

$$\text{k20} \quad = \quad \text{k}(0, \ \text{k1}, \ \text{k2}, \ 0); \tag{30}$$

- And
    - In
        - Statement 30:
            - Nothing

will

- Be
    - Replaced.
        - And:
            - So

a

- Copy
  - Of:

$$k$$

will

- Be
  - Given
    - To:

$$k20.$$

- And
  - If:

$$k1.length == 0 \quad or \quad k2.length == 0,$$

then

- Nothing
  - Will
    - Be:
      - Replaced

in

- Statement 29.
  - Let:

$$[int\ \ int]\ \ k([int\ \ int],\ \ [int\ \ int]); \tag{31}$$

be

- Some
  - Method.

- ▹ Then:
  - If

we

- Write:

$$k20 \quad = \quad k(k1,\ k2);$$

we

- Will
  - Be
    - ▹ Referring:
      - To method 31.
- But:

$$k20 \quad = \quad k[\,](k1,\ k2);$$

is

- Similar
  - To
    - ▹ Statement 28.
      - And:

$$k20 \quad = \quad k[\,](8,\ \ k1[\,],\ \ k2[\,],\ \ 9);$$

is

- Equivalent
  - To.

$$k20 \quad = \quad k(8,\ \ k1,\ \ k2,\ \ 9);$$

- The
  - Interpretation
    - ▹ Of:

$$[\text{int}\ \ \text{int}] \quad k \quad = \quad k1[\,][0]\ \ k2[\,][1];$$

- Is:

```
[int int] k;
if (k1.length < k2.length)
    Prune the right side of: k2[ ][0]
    until: k1[ ][0] and k2[ ][1] are of the same length.
else
    Prune the right side of: k1[ ][0]
    until: k1[ ][0] and k2[ ][1] are of the same length.
for(int i in 0 .. k1.length)
    k += k1[i][0] k2[i][1];
```

- And:

```
[int int int int] k4 = k1[ ] k2, k1[ ] k2[ ], k1[ ][1, 0] k2[ ];
```

is

- Equivalent
  - To:

```
[int int int int] k4 = k1 k2, k1 k2, k1[ ][1, 0] k2;
```

- And:

```
[int int int int] k4 = k1[ ] k2[ ], k1[ ] k2[ ] : (...);
```

is

- Like.

```
[int int int int] k4 = k1[ ] * k2[ ], k1[ ] * k2[ ] : (...);
```

- The
  - Interpretation
    - Of:

$$k = (k1[\,] : (\text{bool-Exp}))[\ ..\ 1];$$

- Is:

“*choose an element of*: k1 *that satisfy*: bool-Exp.”

- Let:

scl

be

- An
  - Instance
    - Of:

[SuperClass],

- And
  - Let:

SubClass *extends*: SuperClass.

- Then
  - We
    - Can
      - Write.

scl = null, new SubClass(), null;

- Let:

```
class ClassOne{
    public int someInt;
    public ClassOne(){...}
    public ClassTwo objReturner(){...}
    public ClassTwo int tupleReturner(ClassOne co){...}
    public boolean boolReturner(ClassOne co){...}
}
```

- And
  - Let:

k12 *and* k13

be

- Instances
  - Of:

[ClassOne],

- And
  - Let:

k14

be

- An
  - Instance
    - Of:

[ClassTwo],

- And
  - Let:

k15

be

- An
  - Instance
    - Of:

[ClassOne ClassTwo],

- And
  - Let:

k16

be

- An
  - Instance
    - Of:

[ClassOne int].

- The
  - Interpretation
    - Of:

k14 = k12[ ].objReturner();

- Is:

```
k14          =    ;
for(int  i  in  0  ..  k12.length)
    if  (k12[i]  !=  null)
        k14    +=    k12[i].objReturner();
    else
        k14    +=    null;
```

- And
  - That
    - Of:

```
k5    =    k12[ ].someInt;
```

- Is:

```
k5          =    ;

for(int i in 0 .. k12.length)

    if (k12[i] != null)

        k5   +=   k12[i].someInt;

    else

        k5   +=   <default-value>;
```

- And
  - That
    - Of:

```
            k16   =   k12[ ].tupleReturner(k12[ ]);
```

- Is.

```
k16             =    ;

for(int i1 in 0 .. k12.length)

    for(int i2 in 0 .. k12.length)

        if (k12[i1] != null && k12[i2] != null)

            k16   +=   k12[i1].tupleReturner(k12[i2]);

        else

            k16   +=   <null-value> <default-value>;
```

- And

- So
  - If:

$$\begin{aligned}&\text{int} && \text{intReturner(ClassOne);}\\ &\text{int} && \text{intReturnerTwo(int);}\\ &\text{ClassOne} && \text{objReturner(int);}\\ &\text{ClassTwo} && \text{objReturnerTwo(ClassOne);}\\ &\text{ClassTwo} && \text{objReturnerThree(ClassOne, ClassOne);}\\ &\text{ClassTwo int} && \text{tupleReturner(ClassOne ClassOne);}\\ &\text{boolean} && \text{boolReturner(ClassOne ClassOne);}\end{aligned} \qquad (32)$$

are

- Methods,
  - And:
    - If

an

- Element
  - Of:

$$\text{k12} \qquad \textit{or} \qquad \text{k13}$$

- Is:

$$\text{null,}$$

- And
  - We

▷ Execute:

$$
\begin{aligned}
\text{k5} &= \text{intReturner(k12[ ]);}\\
\text{k5} &= \text{intReturnerTwo(k12[ ].someInt);}\\
\text{k13} &= \text{objReturner(k12[ ].someInt);}\\
\text{k14} &= \text{objReturnerTwo(k12[ ]);}\\
\text{k14} &= \text{objReturnerThree(k12[ ],\quad k13[ ]);}
\end{aligned}
$$

the

- Corresponding
  - Elements
    - ▷ Of:

k5, k13 *and* k14

will

- Be:

default-value, *null-value* *and* *null-value*

respectively.

- And
  - Similarly,
    - ▷ If:
      - An element

of

- Either:

$$\text{k15[ ][0]} \qquad \textit{or} \qquad \text{k15[ ][1]}$$

- Is:

$$\text{null,}$$

- And
    - We
        - Execute:

$$\text{k16} \quad = \quad \text{tupleReturner(k15[ ]);}$$

the

- Corresponding
    - Element
        - Of:

$$\text{k16}$$

will

- Be:

$$\text{<null-value>} \quad \text{<default-value>.}$$

- And:

$$\text{boolReturner(k15[ ])} \quad == \quad \text{false,}$$

if

- An
    - Element
        - Of
            - Either:

$$\text{k15[ ][0]} \qquad \textit{or} \qquad \text{k15[ ][1]}$$

- Is:

$$\text{null.}$$

- And:

$$(\text{boolReturner}(\text{k15}[\,])\quad(8) == \quad\text{true})\quad == \quad\text{true},$$

if

- At
    - Least
        - Eight
            - Elements of:

$$\text{k15}$$

are

- Not:

$$\text{null,}$$

- And
    - Also
        - Satisfy:
            - Method 32.
- And
    - Similarly,
        - For:

$$\text{boolReturner}(\text{k15}[\,])\quad([8]) == \quad\text{true.}$$

- The
    - Interpretation

▹ Of:

```
b = k12[ ].boolReturner(k12[ ]);
```

- Is.

```
b = true;
for(int i1 in 0 .. k12.length)
    for(int i2 in 0 .. k12.length)
        if (k12[i1] != null && k12[i2] != null){
            if (k12[i1].boolReturner(k12[i2]) == false)
                b = false;
        }
        else
            b = false;
```

- Let:

co1 *and* co2

be

- Objects.
  - Then:

```
[int] k5 = (co1, co2)?;
```

is

- Equivalent

    - To:

```
[int]  k5  =   co1?;
k5        +=  co2?;
```

- And:

```
[int int]  k  =  k15[ ]?;
```

- To:

```
[int int]  k;
for(int  i  in  0  ..  k15.length)
    k  +=  k15[i][0]? k15[i][1]?;
```

- And:

```
int  i  =  k15[ ]?;
```

- To:

```
int  i  =  (k15[0][0],  k15[0][1],  ...  )?;
```

- And:

int i = k15[ ]?; *and* int i = k15?;

are

- Different.
    - Let:

k25

be

- An
  - Instance
    - ▹ Of:

$$[(\text{[int}\quad \text{int]}\quad \text{subset})].$$

- And
  - If:

$$\text{k25} \quad = \quad (\text{enum.subsets[ ]})\text{k};$$

- Then:

$$\text{k25[ ].subset}$$

will

- Store
  - All
    - ▹ Subsets
      - – Of:

$$\text{k},$$

- And
  - The
    - ▹ Order
      - – In all:

$$\text{k25[ ].subsets}$$

will

- Be

- The
  - ▹ Same:
    - – As

that

- In:

k.

- And
  - Similarly,
    - ▹ Using:

enum.subsets[8 ..], enum.subsets[8 .. 80], . . . ,

and

enum.sublists[8 ..], enum.sublists[8 .. 80], ...

- And:

k25 = (enum.perm[8])k;

can

- Be
  - Used
    - ▹ To:
      - – Generate

the

- The
  - Set
    - ▹ Of

- All:

$$\text{8-permutations,}$$

- And:

$$\text{k25} \quad = \quad \text{(enum.comb[8])k;}$$

to

- Generate
  - The
    - Set
      - Of all:

$$\text{8-combinations,}$$

- And
  - Similarly,
    - Using:

$$\text{enum.perm[ ],} \qquad \text{enum.perm[8 ..],} \qquad \ldots,$$

and

$$\text{enum.comb[ ],} \qquad \text{enum.comb[8 ..],} \qquad \ldots.$$

- And:

$$\text{k25} \quad = \quad \text{(enum.sublists[ ])k[ ]} \quad : \quad (\ldots);$$

is

- Equivalent
  - To:

$$[\text{int} \quad \text{int}] \quad \text{k1} \quad = \quad \text{k}[\,] \quad : \quad (\ldots);$$

$$\text{k25} \quad = \quad (\text{enum.sublists}[\,])\text{k1};$$

- And
  - Similarly,
    - ▹ For.

$$\text{k25} \quad = \quad (\text{enum.sublists}[\,])\text{k}[\,], \quad (\text{enum.perm}[\,])\text{k}[\,], \quad \text{k} \quad : \quad (\ldots);$$

- Let:

$$\text{boolean} \qquad \text{boolReturner}([\text{int} \quad \text{int}]);$$

$$\text{boolean} \quad \text{boolReturnerTwo}([\text{int} \quad \text{int}]);$$

be

- Methods.
  - The
    - ▹ Interpretation
      - Of:

$$\text{k25} \quad = \quad (\text{enum.subsets}[\,][\text{boolReturner}])\text{k}[8 \quad .. \quad];$$

- Is:

"*select all subsets of*: $\text{k}[8 \quad .. \quad]$ *that satisfy*: boolReturner."

- And
  - Similarly,
    - ▹ Using:

$$\text{enum.subsets}[8 \quad .. \quad][\text{boolReturner} \quad || \quad \text{boolReturnerTwo}].$$

- The
  - Interpretation
    - ▷ Of:

boolean b = ([8])(enum.sublists[ ][boolReturner])k;

- Is:

"*does*: k *contain exactly*: 8 *non-intersecting sublists that satisfy*: boolReturner,"

- And
  - That
    - ▷ Of:

([0] == 1, [0] == (+)[1], [1] == 8)

- Is:

```
Select a tuple, say: $t_i$, such that: $t_i[0] == 1$.
repeat
    Select a tuple, say: $t_{i+1}$, such that: $t_i[0] == t_{i+1}[1]$.
    $t_i = t_{i+1}$;
until($t_i[1] == 8$).
```

- And so
  - If
    - ▷ We
      - Use:

$$\text{enum.subsets}[8][([0] == 1,\ [0] == (+)[1],\ [1] == 8)]$$

the

- Interpretation
  - Will
    - ▹ Be:

$$\text{“}\textit{select all paths of length}:\quad 8$$

$$\textit{defined by}:\quad ([0] == 1,\ [0] == (+)[1],\ [1] == 8)\text{.”}$$

- And
  - In:

$$\text{k25} = (\text{enum.sublists}[\,][\text{path-Exp}])\text{k};$$

- If:

$$\text{k}[\text{i}]$$

was

- Chosen
  - As
    - ▹ The:
      - Initial-element,

then

- Only:

$$\text{k}[\text{i} + 1]$$

can

- Be
  - Chosen:

        ▷ As

the

- Next
    - Element.
        ▷ But:
            - It

will

- Not
    - Be
        ▷ The:
            - Case,

if

- We
    - Use:

enum.subsets[ ][path-Exp].

- And
    - We
        ▷ Can
            - Use:

(intReturner([2], [0]) in k5, [0, 2] > (+)[1, 4], [1, 4] < 0 0)

or

enum.sublists[ ][([0].<method>(<parameters>) == 0, . . .)]

or

enum.multi[i1][i2][i3][path-Exp && boolReturner] *or*

enum.multisubsets[ ][. . .].

- The
  - Interpretation
    - ▹ Of:

[int]   k5   =   (enum.sublists[ ][boolReturner])k;

- Is:

"*find the indices of all non-intersecting sublists of*:   k

that satisfy: boolReturner."

- And
  - We
    - ▹ Do:
      - Not

allow

- Statements
  - Like:

k5   =   (enum.<subsets-or-perm-or-comb>[ ][boolReturner])k;

- And:

(8)(&&&)(enum.subsets[9 ..][boolReturner || boolReturnerTwo])

is

- Equivalent
  - To:

$$(\&\&\&)(8)(\text{enum.subsets}[9\ ..\ ][\text{boolReturner}\ \ ||\ \ \text{boolReturnerTwo}]).$$

- Let:

$$\text{state}$$

be

- A keyword.
    - And
        - ▷ Let:

$$[\text{int}\ \ \text{int}]\quad k \quad = \quad \ldots;$$

- And
    - We
        - ▷ Write:

```
k.transient    =    (t){
                          k[state][0]    +=    t[0];
                          k[state][1]    +=    t[1];
                          t              =     k[state];
                    };
```

- And
    - Execute:

$$\begin{aligned}
&\text{k.state} && = && \text{<some-int>;}\\
&\text{k.return} && = && 10 \quad 20;\\
&\text{k+;} && && \\
&\text{int} \quad \text{int} \qquad \text{t2} && = && \text{k.return;}
\end{aligned}$$

- And
  - When:

$$\text{k.return} \quad = \quad 10 \quad 20;$$

is

- Executed,
  - The
    - ▷ Parameter:
      - Of

that

- Block
  - Will
    - ▷ Be
      - Assigned:

$$10 \quad 20,$$

- And
  - When:

$$\text{k+;} \tag{33}$$

is

- Executed,
    - The
        - Statements:
            - In

that

- Block
    - Will
        - Be:
            - Executed,
- And
    - The
        - Value
            - In:

t

will

- Be
    - Saved,
        - And:

k.state

will

- Be
    - Incremented,
        - And
            - When:

int int t2 = k.return;

is

- Executed,
    - The
        - ▹ Value
            - In:

t

will

- Be
    - Given
        - ▹ To:

t2.

- And
    - If:

k+;

k+;

first

- That block
    - Will
        - ▹ Be executed
            - For:

k[k.state]

using

- The
    - Value
        - ▹ In:
            - – The parameter,
- And
    - Then
        - ▹ The:
            - – Value

in

- The
    - Parameter
        - ▹ Will:
            - – Be saved,
- And:

k.state

will

- Be incremented,
    - And
        - ▹ The:
            - – Execution

will

- Proceed.
    - Note that,
        - ▹ If
            - – We write:

$$[\text{int} \quad \text{int}] \quad \text{k} \quad = \quad (\text{t})\{\ldots\};$$

- Then:

$$\text{t}$$

will

- Be
  - Type:

$$\text{int} \quad \text{int},$$

- Since:

$$\text{k}$$

is

- Of
  - Type:

$$[\text{int} \quad \text{int}].$$

- And
  - So
    - ▹ These:
      - Blocks

can

- Have
  - Only
    - ▹ One:
      - Parameter.
- The

- Interpretation
  - ▷ Of:

$$k-; \tag{34}$$

is

- Similar,
  - Except
    - ▷ That:
      - After executing

that

- Block:

k.state

will

- Be
  - Decremented.
    - ▷ The
      - Interpretation of:

k;

is

- Similar,
  - Except
    - ▷ That:
      - There

will

- No

- Change
  - In:

$$\text{k.state.}$$

- And
  - If:

$$\text{k} + *; \qquad (35)$$

then

- That block
  - Will
    - Be executed
      - From:

“*the* k.stateth *element*”

to

- The last.
  - And
    - Similarly,
      - Define:

$$\text{k} - *; \qquad (36)$$

- And
  - After
    - Executing
      - Statements 35 or 36:

$$\text{k.state} == \text{k.length} \qquad \textit{or} \qquad \text{k.state} == -1$$

respectively.

- And
  - If:

$$\text{k.state} < 0, \qquad \text{k.length} <= \text{k.state} \qquad \textit{or} \qquad \text{k.length} == 0,$$

- And
  - We
    - ▹ Execute:

$$\text{k}-; \qquad \text{k} - *; \qquad \text{k}; \qquad \text{k}+; \qquad \textit{or} \qquad \text{k} + *; \quad (37)$$

then

- That
  - Block
    - ▹ Will:
      - Not

be

- Executed.
  - And
    - ▹ So:
      - There

there

- Will
  - Be
    - ▹ No:
      - Change

in

- Parameter
    - Value,
        - ▷ And
            - Also in:

k.state,

- And
    - Also
        - ▷ No:
            - Exception.
- And
    - If
        - ▷ Statements 33 and 34:
            - Throws

an

- Exception,
    - The
        - ▷ Value
            - Of:

k.state

will

- Remain
    - Unchanged.
        - ▷ And:
            - If statement 35 or 36

throws

- An
    - Exception,
        - ▹ The
            - Value of:

k.state

will

- Be
    - The
        - ▹ Ordinal:
            - Position

of

- The
    - Element
        - ▹ From:
            - Where

the

- Exception
    - Was:
        - ▹ Thrown.
- And
    - If
        - ▹ This:
            - Block

is

- Undefined,

- Then:

$$\text{k.transient} \quad == \quad \text{null}. \tag{38}$$

- And
  - So:

$$\text{k.transient} \quad = \quad \text{null};$$

can

- Be
  - Used
    - ▷ To:
      - Remove it,
- And:

$$\text{k.transient} \quad = \quad (\text{t1})\{\ldots\};$$

to

- Redefine it.
  - And when
    - ▷ Expression 38:
      - Holds,
- And
  - We
    - ▷ Execute:
      - Statements 37,

there

- Will
  - Be

        ▹ An:
            – Exception.

- And
    - We
        ▹ Can:
            – Say

that,

- If:

$$\text{boolean} \quad b \quad = \quad \text{k.transient?;}$$

- Then:

$$b \;==\; \text{true,}$$

if

- The
    - Value
        ▹ In:
            – The parameter

is

- The
    - Result
        ▹ Of:
            – Execution

of

- That block.
    - But
        ▹ Since:

- It

will

- Not
  - Be
    - So:
      - Useful,

we

- Do
  - Not
    - Allow:
      - It.
- Assume
  - That:

$$\mathscr{A}$$

holds

- The
  - Address:
    - Of

the

- Right
  - Hand
    - Side
      - Of.

$$\mathrm{k} = \ldots;$$

- Then
    - The three
        - Locations
            - After:

$$\mathscr{A}$$

will

- Hold
    - The
        - Addressees:
            - Of

the

- Parameter
    - Of
        - That:
            - Block,
- And:

$$\text{k.state} \qquad \textit{and} \qquad \text{k.transient.}$$

- And
    - So
        - If:

$$\text{k1} = \text{k2};$$

there

- Will
    - No

      ▹ Change
        – In:

$$\text{k1.state} \qquad \textit{and} \qquad \text{k1.transient.}$$

- But
  - If:

$$\text{k1.transient} = \text{k2.transient;}$$

- Then:

$$\text{k1.transient}$$

will

- Be
  - A copy
      ▹ Of:

$$\text{k2.transient.}$$

- And
  - We
      ▹ Can
        – Write:

$$\text{void someMethod([int int] k)\{ k+; \}}$$

- And:

$$\begin{aligned} \text{k1.transient} &= (\text{t})\{\ldots\}; \\ \text{k2.transient} &= (\text{t})\{\ldots\}; \\ \text{k1.state} &= \ldots; \end{aligned}$$

```
k2.state             =    k1.state;
k1                   =    k;
k2                   =    k;
k1+;
k2−;
```

instead

- Of:

```
k.transient          =     (t){...};
k.transient          +=    (t){...};
k.transient[0].state =     ...;
k.transient[1].state =     k.transient[0].state;
k.transient[0]+;
k.transient[1]−;
```

- And
    - If
        - We:
            - Did

not

- Initialize:

k.state,

it

- Will
  - Be
    - ▹ Initialized
      - To:

"*default-value*,"

- And
  - Similarly,
    - ▹ For:
      - The parameter.
- And:

$$\begin{array}{llcl}
[\text{int} \;\; \text{int} \;\; \text{int}] & \text{k3} & = & \ldots; \\
[\text{int}] & \text{k5} & = & (-)\text{k3}[\,][0]; \\
[\text{int} \;\; \text{int}] & \text{k2} & = & (-)\text{k3}[\,][1, \;\; 0]; \\
[\text{int} \;\; \text{int} \;\; \text{int}] & \text{k19} & = & (-)\text{k3}[\,];
\end{array}$$

can

- Be
  - Used
    - ▹ For:
      - Reversing,
- And:

$$\begin{array}{llcl}
[\text{int} \quad \text{int} \quad \text{int}] & \text{k3} & = & \ldots; \\
[\text{int}] & \text{k5} & = & (0)\text{k3}[\,][0]; \\
[\text{int} \quad \text{int}] & \text{k2} & = & (0)\text{k3}[\,][1, \quad 0]; \\
[\text{int} \quad \text{int} \quad \text{int}] & \text{k19} & = & (0)\text{k3}[\,];
\end{array}$$

for

- Removing
  - Reptitions
    - Without:
      - Changing

the

- Order.
  - And
    - If:

$$\text{k.length} \quad == \quad 0,$$

- Then:

$$(-)\text{k} \qquad \textit{and} \qquad (0)\text{k}$$

will

- Not
  - Throw
    - Any:
      - Exception.
- And

- If:

$$\text{boolean} \quad b \quad = \quad (0)k;$$

- Then:

$$b \; == \; \text{true},$$

if

- There
  - Is
    - ▹ No repetition
      - In:

$$k.$$

- And:

$$\begin{aligned}
&k \quad = \quad (0)k[\,][0] \;*\; (-)k[\,][1] \;:\; (\ldots);\\
&k5 \quad = \quad (0)k6 \;\; \&\& \;\; 10, \quad (-)k6 \;\; \&\& \;\; 10;\\
&(0)(-)k+;\\
&(-)(0)k+;
\end{aligned}$$

is

- Equivalent
  - To:

```
[int int]   k36   =   (0)k,   k37   =   (−)k;
k                 =   k36[ ][0]  ∗  k37[ ][1]  :  (. . .);
k5                =   ((0)k6)  &&  10,   ((−)k6)  &&  10;
((0)((−)k))+;
((−)((0)k))+;
```

- And
  - If:

```
[("colOne")int  int  ("stringVar")string]   k18;
```

- Then:

```
k18[ ]["colOne"]
```

can

- Be
  - Used
    - ▷ Instead
      - Of:

```
k18[ ][0].
```

- And
  - If:

```
@((<)[0],  (>)[1])
[int  int]   k;
```

- Then:

k

will

- Always
  - Be
    - Kept:
      - Sorted-accordingly.
- And
  - Similarly,
    - For:

```
@((0)[ ])
[int  int]    k;
```

- And
  - We
    - Do:
      - Not

allow

- Lists
  - In:
    - Tuples.
- And so
  - We
    - Cannot
      - Write.

$$\text{int} \quad [\text{int}] \quad \text{int} \qquad t;$$

- Let:

arrA, arrB, arrC *and* arrD

be

- Instances
  - Of:

$$\text{int} \quad \text{int} \quad [\,][\,].$$

- The
  - Interpretation
    - ▹ Of:

$$\text{int} \quad \text{int} \quad [\,][\,][\,][\,] \quad \text{arrF} \quad = \quad \text{arrA}[\,][\,][0] \;*\; \text{arrB}[\,][\,][1];$$

- Is.

```
int int [ ][ ][ ][ ] arrF = . . . ;
for(int i in 0 .. arrA.length)
    for(int i2 in 0 .. (arrA[i]).length)
        for(int j in 0 .. arrB.length)
            for(int j2 in 0 .. (arrB[j]).length)
                arrF[i][i2][j][j2] = arrA[i][i2][0] arrB[j][j2][1];
```

- Let:

$$\text{int} \qquad\qquad \text{methodOne(int} \quad \text{int);}$$

```
int      methodTwo(int  int,    int  int);

int              methodThree(int,  int);          (39)
```

be

- Methods.
  - The
    - ▷ Interpretation
      - Of:

```
int [ ][ ]   arrH    =    methodOne(arrA[ ][ ]);
```

- Is:

```
int [ ][ ]   arrH   =   . . . ;

for(int  i  in  0  ..  arrA.length)

    for(int  j  in  0  ..  (arrA[i]).length)

        arrH[i][j]   =   methodOne(arrA[i][j]);
```

- And
  - That
    - ▷ Of:

```
int [ ][ ][ ][ ]   arrK    =    methodTwo(arrA[ ][ ],  arrB[ ][ ]);
```

- Is:

```
int [ ][ ][ ][ ]   arrK            =   . . . ;

for(int  i  in  0  ..  arrA.length)
```

```
        for(int i2 in 0 .. (arrA[i]).length)
            for(int j in 0 .. arrB.length)
                for(int j2 in 0 .. (arrB[j]).length)
                    arrK[i][i2][j][j2] = methodTwo(arrA[i][i2], arrB[j][j2]);
```

- Or
  - In:

```
int [ ][ ][ ][ ] arrK = methodThree(arrA[ ][ ][0], arrB[ ][ ][1]);
```

the

- Compiler
  - Can
    - ▷ Note:
      - That

the

- Range
  - Operator
    - ▷ Has:
      - Been used,
- And:

$$\dim(arrK) == \dim(arrA) + \dim(arrB),$$

- And
  - So
    - ▷ Use:
      - Method 39,

- And
  - Similarly,
    - For:

int [ ][ ] arrH = methodThree(k1[ ][0], k2[ ][1]);

- And
  - In:

[int] k5 = methodThree(arrA[ ][ ][0], arrB[ ][ ][1]);

the

- Compiler
  - Can
    - Note:
      - That

the

- Range
  - Operator
    - Has:
      - Been used,
- And:

arrA[ ][ ][0] *and* arrB[ ][ ][1]

can

- Be
  - Seared
    - To:
      - Lists,

- And
    - So
        - ▹ Use:
            - Method 39,
- And
    - In:

int [ ][ ] arrH = someMethod(arrA[ ][ ][0]);

the

- Compiler
    - Can
        - ▹ Note:
            - That

the

- Range
    - Operator
        - ▹ Has:
            - Been used,
- And
    - There
        - ▹ Is
            - No:

int someMethod(int);

- And
    - So
        - ▹ Signal:

            - An error.

- And
    - So
        - In:
            - General,

we

- Do
    - Not
        - Allow:
            - Methods

that

- Returns:
    - A composite
        - Data
            - Type

to

- Be
    - Used
        - In:
            - Transformation.

- Let:

```
class  ClassThree{
    public   int   someInt;
    public  ClassThree(){...}
```

```
        public ClassFour objReturner(){...}

        public ClassFour int tupleReturner(ClassThree ct){...}

    }
```

- And
  - Let:

arrL *and* arrM

be

- Instances
  - Of:

ClassThree [ ],

- And
  - Let:

arrN

be

- An
  - Instance
    - ▹ Of:

ClassThree ClassThree [ ],

- And
  - Let:

```
int              intReturner(ClassThree);
int              intReturner(int);
ClassThree       objReturner(int);
ClassFour        objReturner(ClassThree);
ClassFour        objReturner(ClassThree, ClassThree);
ClassFour int    tupleReturner(ClassThree ClassThree);
```

be

- Methods.
  - Then
    - ▹ We:
      - Can

give

- An
  - Interpretation
    - ▹ For
      - Expressions:

```
intReturner(arrL[ ]),                intReturner(arrL[ ].someInt),
objReturner(arrL[ ].someInt),        objReturner(arrL[ ]),
objReturner(arrL[ ], arrM[ ]),       tupleReturner(arrN[ ]),
arrL[ ].objReturner(),               arrL[ ].someInt,
arrL[ ].tupleReturner(arrL[ ]),      arrN[ ]?
```

like

- That
    - Which
        - We:
            - Did

for

- Lists.
    - And:

```
int [ ] arr   =   ( 10, 20, 30, 40 );

arr           =   ( 0, 1 );
```

is

- Equivalent
    - To:

```
int [ ] arr   =   ( 10, 20, 30, 40 );

arr           =   new int[2];

arr[0]        =   0;

arr[1]        =   1;
```

- And
    - We
        - Can:
            - Say

that,

- If:

$$\text{arrB}$$

points

- To:

$$((10\quad 10,\qquad 20\quad 20,\qquad 30\quad 30),\qquad (100\quad 100,\qquad 200\quad 200,\qquad 300\quad 300)),$$

- And
  - We
    - ▷ Execute:

$$\text{arrA}\quad =\quad \text{arrB}[\,][\,]\quad :\quad (\text{arrB}[\,][\,][0]\ \ <=\ \ 200);$$

- Then:

$$\text{arrA}$$

will

- Point
  - To:

$$(\ \ (\ \ ),\qquad (\ \ 200\quad 200,\qquad 300\quad 300\ \ )\ \ ).$$

- And
  - So
    - ▷ Define.

$$\text{arrA}\qquad \%=\qquad \text{arrB}[\,][\,]\quad :\quad (\ldots);$$

$$\text{arrA}\qquad \%=\qquad 200\quad 200,\qquad 300\quad 300;$$

- But
    - Since
        - ▷ It:
            - Will complicate,

we

- Say
    - That:
        - ▷ If

a

- Condition
    - Part
        - ▷ Is:
            - Applicable

to

- An
    - Array,
        - ▷ Then:
            - That array

will

- Be
    - Seared
        - ▷ To:
            - A list.
- And
    - So:

$$\text{arrA} \quad = \quad \text{arrB}[\,][\,] \quad : \quad (\ldots);$$

should

- Be
  - Rewritten
    - As:

$$[\text{int} \ \ \text{int}] \quad \text{k} \quad = \quad \text{arrB}[\,][\,] \quad : \quad (\ldots);$$

- And
  - We
    - Do:
      - Not

allow

- Statements
  - Like:

$$\text{arrA} \qquad \%= \qquad 200 \ \ 200, \qquad 300 \ \ 300;$$

$$\text{arrA}[\,][\,] \qquad = \qquad \text{arrB}[\,][\,];$$

- But
  - We
    - Can
      - Write:

$$\text{arrE} \qquad = \qquad \text{arrA}[\,][\,][0] \ \ \text{arrB}[\,][\,][1, \ \ 0] \ \ \text{arrA}[\,][\,][1];$$

$$\text{arrA}[\,][\,][0]{+}{+} \qquad : \qquad (\ldots);$$

$$\text{arrA}[\,][\,][0] \quad += \quad \text{i} \qquad : \qquad (\ldots);$$

$$i = (\text{arrA}[0][\,][0] \quad : \quad (\ldots)).\text{max};$$

$$i = (\text{arrA}[0][\,][0] \quad : \quad (\ldots)).\text{min};$$

$$i = (\text{arrA}[0][\,][0] \quad : \quad (\ldots)).\text{sum};$$

$$i = \text{arr.max} + \text{arr.min} + \text{arr.sum};$$

- Where:

$$\text{arrE}$$

is

- An
  - Instance
    - ▷ Of:

$$\text{int int int int } [\,][\,].$$

- The
  - Interpretation
    - ▷ Of:

$$\text{arrA} = \text{arrB}[\,][\,] + (8 \quad 8); \tag{40}$$

- Is:

```
for(int int ae in arrB)
    The corresponding element of: arrA
    is: ae + (8 8).
```

- But
    - We
        - ▹ Do not
            - Allow:

$$\text{arrA} = \text{arrB}[\,][\,] + \text{arrC}[\,][\,]; \tag{41}$$

- Or
    - In
        - ▹ Statements:
            - Like statement 40

only

- One
    - Range
        - ▹ Operator:
            - Can

be

- Juxtaposed
    - To
        - ▹ One:
            - Array.
- And
    - So
        - ▹ We:
            - Do

allow

- Statements

    - ◦ Like:
        - ▹ Statement 41.

- And
    - ◦ Define:

in *and* !in,

- And
    - ◦ All
        - ▹ Other:
            - – Boolean-operations,

- And:

the for-all-statement *and* *the there-exists-statement*

like

- That
    - ◦ Which
        - ▹ We:
            - – Did

for

- Lists.
    - ◦ And:

([arrA] == [arrB]) == true,

- If:

arrA *and* arrB

points

- To
  - The
    - Same:
      - Location.
- And:

$$\text{arrA} = [\text{arrB}]; \qquad \textit{and} \qquad \text{arrA} = \text{arrB};$$

are

- Equivalent.
  - And:

$$[\text{arrA}][\,][\,] \qquad \textit{and} \qquad [\text{arrA}][\,][\,][0]$$

are

- Equivalent
  - To:

$$\text{arrA}[\,][\,] \qquad \textit{and} \qquad \text{arrA}[\,][\,][0]$$

respectively.

- And
  - We
    - Can
      - Write:

$$[\text{int}\ \ \text{int}] \quad \text{k} = \text{arrB}\ \ \&\&\ \ \text{arrC};$$

- But
  - Not:

$$\text{arrA} = \text{arrB}\ \ \&\&\ \ \text{arrC};$$

- And
    - Similarly,
        - ▷ For:
            - Other operators.
- The
    - Interpretation
        - ▷ Of:

$$[\text{int} \quad \text{int}] \quad \text{k} \quad = \quad \text{arrB} \;<\; \text{arrA};$$

- Is:

“*find all indices of*: arrB *in*: arrA.”

- And
    - If:

$$[\text{int} \quad \text{int} \quad \text{int} \quad \text{int}] \quad \text{k4} \quad = \quad \text{arrB} \;<\; \text{arrA};$$

- Then:

$$\text{k4}[\,][0, \;\; 1]$$

will

- Hold:

“*the left-upper-bounds*,”

- And:

$$\text{k4}[\,][2, \;\; 3]$$

will

- Hold:

"*the right-lower-bounds* $+$ (1 1)."

- And
    - Similarly,
        - ▷ For:
            - Other dimensions.
- Let:

boolean boolReturner(int [ ][ ]);

be

- Some
    - Method,
        - ▷ And
            - Let:

arrH

be

- An
    - Instance
        - ▷ Of:

int [ ][ ].

- The
    - Interpretation
        - ▷ Of:

k4 $=$ (enum.subarrays[8][8])arrH;

- Is:

$$\text{“}\textit{find the bounds of all}: \quad 8 \times 8 \quad \textit{subarrays}$$

$$\text{“}\textit{that satisfy}: \quad \text{boolReturner.”}$$

- And
  - We
    - ▹ Can:
      - Give

a

- Description
  - Like
    - ▹ That:
      - Which

we

- Did
  - For:
    - ▹ Lists.
- And
  - We
    - ▹ Do:
      - Not

allow

- Statements
  - Like:

$$\text{arrA} \quad = \quad \text{arrB}(\text{arrC}, \ \ \text{arrD});$$

- And

    - If:

$$\text{arrH.transient} \quad = \quad (\text{t})\{\ldots\};$$

- Then:

t

will

- Be
    - Of
        - ▹ Type:

int,

- And:

arrH.state

will

- Be
    - Of
        - ▹ Type:

int int.

- And
    - If:

$$\text{arrH}-+;$$

then

- After
    - Executing

        - ▷ That:
            - Block,

the

- Value
    - Of:

$$\text{arrH.state}[0]$$

will

- Be
    - Decremented,
        - ▷ And:

$$\text{arrH.state}[1]$$

will

- Be
    - Incremented.
        - ▷ The
            - Interpretation of:

$$\text{arrH}{+}{+};$$

is

- Similar,
    - Except
        - ▷ That:
            - After executing

that

- Block,

- ◦ Both:

$$\text{arrH.state}[0] \quad \textit{and} \quad \text{arrH.state}[1]$$

will

- Be
    - ◦ Incremented.
        - ▹ The
            - – Interpretation of:

$$\text{arrH+!;}$$

is

- Similar,
    - ◦ Except
        - ▹ That:
            - – After executing

that

- Block:

$$\text{arrH.state}[0]$$

will

- Be
    - ◦ Incremented,
        - ▹ And:

$$\text{arrH.state}[1]$$

will

- Remain

- ○ Unchanged.
  - ▷ And
    - – Define.

$$\text{arrH;} \qquad \text{arrH}--; \qquad \text{arrH}+-;$$

$$\text{arrH}-!; \qquad \text{arrH!}-; \qquad \text{arrH!}+;$$

- The
  - ○ Interpretation
    - ▷ Of:

$$\text{arrH!}+*;$$

- Is:

$$\text{arrH!}+;$$

$$\text{arrH!}+;$$

$$\vdots$$

- And
  - ○ That
    - ▷ Of:

$$\text{arrH}++*;$$

- Is:

arrH++;

arrH++;

$\vdots$

- And
  - Similarly,
    - For:
      - All

other

- Combinations.
  - And
    - Define:

$$---, \qquad --+, \qquad -+-, \qquad -++,$$

$$+--, \qquad +-+, \qquad ++-, \qquad +++,$$

- And
  - Also
    - Using:

$$! \qquad \textit{and} \qquad *$$

- Like:

$$!++, \qquad !+!, \qquad !+!*, \qquad \ldots$$

for

- Three

        - Dimensional:
            - Arrays,
- And
    - Similarly,
        - For:
            - Other dimensions.
- And
    - We
        - Can:
            - Give

a

- Description
    - Like
        - That:
            - Which

we

- Did
    - For:
        - Lists.
- And
    - We
        - Can
            - Write:

$$\text{arrH} \quad = \quad ;$$

to

- Remove
  - All
    - ▹ Elements,
      - And:

$$(0)\text{arrH}$$

to

- Remove
  - Repetitions,
    - ▹ And:

$$(-)\text{arrH}$$

to

- Generate
  - The:
    - ▹ Transpose.
- Assume
  - That,
    - ▹ We have
      - Written:

```
public    int     i,    j,    m;

private   [int]   stack           for           i,  j;
```

in

- The:

    - Class
        - Body.

- Then
    - If:

i.push;

⋮

i.pop;

the

- Value
    - In:

i

will

- Be
    - First
        - Pushed
            - Into:

stack,

without

- Changing
    - The
        - Value
            - In:

i,

- And
  - Later
    - ▹ An element
      - From:

stack

will

- Be
  - Popped
    - ▹ Into:

i.

- And
  - Similarly,
    - ▹ If:

```
i.append;

i.push;

j.append;

j.push;

⋮

j.pop;

i.pop;
```

the

- Value
  - In:

i

will

- Be
  - First
    - ▹ Appened
      - To:

stack.

- And
  - So:

stack

will

- Simultaneosuly
  - Act
    - ▹ As:

a stack *and* *a queue*

- For:

i *and* j.

- But:

m.push; m.pop; *and* m.append;

will

- Not
  - Compile,
    - ▹ Since:
      - We

did

- Not
  - Associate
    - ▹ Any
      - Stack with:

m.

- And
  - If
    - ▹ We:
      - Want

to

- Lock

$$i$$

from

- Being
  - Pushed,
    - ▷ We
      - Write:

$$\text{i.push?} = 2;$$

- And
  - If:

$$\text{i.push?} == 2;$$

- And
  - We
    - ▷ Execute:

$$\text{i.push;}$$

the

- Program
  - Will
    - ▷ Continue:
      - As

if

- Nothing
  - Happened.
    - ▷ And:

- We

can

- Write:

$$i.push? = 1;$$

for

- Unlocking.
  - And
    - ▷ Similarly,
      - For:

popping *and* *appending.*

- Let:

$$i$$

be

- Some
  - Field.
    - ▷ Then:
      - To

avoid

- Complications,
  - We
    - ▷ Say that:
      - If

we

- Declare

- A stack
  - ▹ For it,
    - – Both:

i

its

- Stack
  - Should:
    - ▹ Have

the

- Same
  - Scope.
    - ▹ And so
      - – If:

```
class  ClassOne{
    public   int   i;
    public  ClassOne(){}
}
```

then

- We
  - Cannot
    - ▹ Write.

```
class  ClassTwo{
    public    ClassOne    c1;
    public    int             stack1    for    c1.i;
    public  ClassTwo(){...}
    public  void  voidReturner(){
        [ClassOne]    stack2          for    c1;
        ⋮
    }
}
```

- Let.

```
class  ClassThree{
    public    int      i;
    public    [int]     stack     for     i;
    ⋮
}
```

- Then
    - We
        - ▷ Can

– Write:

```
ClassThree     c3     =     ...;

c3.i.push;

c3.stack               +=     10;
```

- And
  - If:

```
class  ClassFour{

    public          int        i;

    protected      [int]      stack     for     i;

    ⋮

}
```

- Then:

i

can

- Only
  - Be:

pushed *or* *popped* *or* *appended*

- Inside:

ClassFour

- And
  - In:
    - Subclasses.
- And
  - Similarly,
    - For:

```
class ClassFive{
    public int i;
    private [int] stack for i;
    ⋮
}
```

- And:

```
(i, j).append;
(i j).push;
⋮
(j, i).pop;
```

- To:

i.append;

j.append;

i.push;

j.push;

⋮

j.pop;

i.pop;

- And
  - To
    - ▹ Avoid
      - – Statements like:

$$\text{int} \quad j \quad = \quad \text{i.pop} \; + \; \text{i.push;}$$

we

- Say
  - That:

i.push, i.pop *and* i.append

cannot

- Be
  - A part
    - ▹ Of:
      - – Other statements.

- Let:

$$c6$$

be

- An
  - Instance
    - ▷ Of.

```
class ClassSix{
    int intReturner() for pop;
    public ClassSix(){}
    private int intReturner(){...};
}
```

- Then
  - If:

$$int \quad i \quad = \quad c6.pop;$$

we

- Will
  - Be
    - ▷ Referring:
      - To

the

- Get

- Property:

pop.

- But
  - If:

c6.pop;

we

- Will
  - Be
    - Poping:

c6

from

- Its
  - Stack.
    - And
      - So:

push, pop *and* append

need

- Not
  - Be:
    - Keywords.
- And
  - Local
    - Variables:
      - Can also

have

- Such
    - Stacks,
        - ▹ And:
            - As

before,

- The
    - Scope
        - ▹ Of:
            - Those variables,
- And
    - That
        - ▹ Of:
            - Their stacks

should

- Be
    - The
        - ▹ Same.
            - Let:

```
int      i,  i2,  j;

[int]    list1      for      i.push,      j.pop;

[int]    list2      for      i.push,      j.push,    i.append;

[int]    list3      for      i.append,    i2,        i.pop;
```

- And
  - We
    - ▷ Execute:

```
i.push;
```

the

- Value
  - In:

i

will

- Be
  - Pushed
    - ▷ Into
      - – Both:

list1 *and* list2.

- And
  - When
    - ▷ We
      - – Execute:

i.pop;

an

- Element
  - From:

list3

will

- Be
  - Removed.
    - But:

j.append;

will

- Not
  - Compile.
    - And:
      - We

do

- Not
  - Allow:

```
int     i;

[int]   list1      for      i.pop;

[int]   list2      for      i.pop;
```

- Or
    - Popping
        - ▷ Can:
            - – Only

be

- Done
    - From
        - ▷ One:
            - – List.
- And
    - We
        - ▷ Can:
            - – Give

a

- Similar
    - Description
        - ▷ Like:
            - – That

which

- We
    - Did:
        - ▷ Earlier.
- And
    - There
        - ▷ Will:
            - – Be

an

- Exception,
    - If
        - ▹ We:
            - Try

to

- Pop
    - From
        - ▹ An:
            - Empty-stack.
- Let:

cs

be

- An
    - Instance
        - ▹ Of:

```
class ClassSeven{
    private                int     i;
    private   static       [int]   staticList;
    private                [int]   list1   for   i.pop;
    protected              [int]   list2;
    public                 [int]   list3;
    list1                                  for   inbox1;
    list2                                  for   inbox2;
    list3                                  for   inbox3;
    staticList                             for   staticInbox;
    public  ClassSeven(){...}
    ⋮
}
```

- And
  - We
    - ▹ Execute:

```
cs.inbox1              =  10;
cs.inbox2              =  20;
cs.inbox3              =  30;
ClassSeven.staticInbox =  40;
```

- Then:

10

will

- Be
  - Appended
    - To:

list1 *of*: cs,

- And
  - So
    - Forth.
      - Let:

```
public class ClassEight{
    public ClassEight(){...}
    public int someMethod(int i){...}
    public int someMethod(int i, int j){...}
    public void someMethod(int i){...}
}
```

- And
    - Let:

```
public class ClassNine{
    public ClassNine(){...}
    public int intReturner(int i){...}
}
```

- And
    - Let:

```
public class ClassTen{
    public ClassTen(){...}
    public void voidReturner(int i){...}
}
```

- And
    - Let:

```
public  class  ClassEleven{
    protected   [int]          k5;
    protected   [int]          k6;
    protected   [ClassEight]   pool1;
    protected   [ClassNine]    pool2;
    protected   [ClassTen]     pool3;
    k5          for          inbox;
    k5          for          (k6;  pool1;  someMethod);
    k5          for          (k6;  pool2;  intReturner);
    k5          for          (;  pool3;  voidReturner);
    public  ClassEleven(){...}
    ⋮
}
```

- And
  - We
    - ▹ Execute:

do[pool1];

then

- All
  - Idle

        ▷ Objects
            – In:

$$\text{pool1}$$

will

- Fetch
    - Elements
        ▷ From:

$$\text{k5}$$

- Using:

$$\text{int} \quad \text{someMethod(int);} \tag{42}$$

- And
    - The
        ▷ Result:
            – Will

be

- Stored
    - In:

$$\text{k6,}$$

until

- We
    - Execute.

$$\text{do![pool1];}$$

- Note that,

    - The compiler
        - ▹ Can:
            - Understand

that,

- Method 42
    - Should
        - ▹ Be:
            - Used,

since

- The
    - Type
        - ▹ Of:
            - Its

only

- Parameter,
    - And
        - ▹ That:
            - Of

the

- Elements
    - Stored
        - ▹ In:

k5

are

- The

- Same,
  - ▹ And
    - Since:

k6

can

- Receive
  - The
    - ▹ Values:
      - Returned

by

- Method 42.
  - And:

```
do[pool1, pool2];
```

is

- Equivalent
  - To:

```
do[pool1];
do[pool2];
```

- And
  - Similarly,
    - ▹ For.

```
                    do![pool1,  pool2];
```

- Let:

```
public  class  ClassTwelve{
    ⋮
}
```

- And
  - Let:

```
public  class  ClassThirteen{
    protected      [ClassTwelve]      k27;
    k27                for                inbox;
    public  ClassThirteen(){...}
    ⋮
}
```

- And
  - Let:

```
public class ClassFourteen{
    public ClassFourteen(){...}
    public void voidReturner(ClassTwelve ct){...}
    ⋮
}
```

- And
  - Let:

```
class ClassFifteen{
    protected    [ClassTwelve]      k27;
    k27          for                inbox;
    protected    [ClassThirteen]    pool1;
    protected    [ClassFourteen]    pool2;
    (; pool1; inbox;),
    (; pool2; voidReturner; bool-Exp)
                                    for     k27;
    public ClassFifteen(){...}
    ⋮
}
```

- And
    - We
        - ▷ Execute:

do[k27];

a

- Clone
    - Of
        - ▷ All
            - Elements in:

k27

will

- Be
    - Put
        - ▷ Into:

pool1[ ].inbox,

if

- They
    - Satisfy:

pool1[ ]? == true,

- And
    - Also
        - ▷ Given
            - To:

pool2[ ].voidReturner(ClassTwo);

if

- They
  - Satisfy:

$$\text{bool-Exp} \qquad \textit{and} \qquad \text{pool2}[\,]? \;==\; \text{true},$$

until

- We
  - Execute.

$$\text{do!}[\text{k27}];$$

- Assume
  - That
    - ▷ We
      - Had written:

$$(0)(;\;\; \text{pool1};\;\; \text{inbox};),\quad (;\;\; \text{pool2};\;\; \ldots;)\quad \text{for}\quad \text{k27};$$

- And
  - If:

$$\text{obj}$$

was

- Put
  - Into:

$$\text{pool1}[<\text{some-int}>].\text{inbox},$$

then

- All
  - Other

      - ▹ Elements
        - In:

$$\text{pool1}$$

will

- Be
  - Skipped.
    - ▹ But:
      - It

will

- Be given
  - To
    - ▹ All elements
      - In:

$$\text{pool2.}$$

- And
  - If
    - ▹ We
      - Had written:

$$(0)(;\ \ \text{pool1};\ \ \ldots;),\qquad(0)(;\ \ \text{pool2};\ \ \ldots;)\qquad\text{for}\qquad\text{k27};$$

- Then:

$$\text{obj}$$

will

- Be
  - Given

▹ To:
  - Exactly

one

- Element
  - Of
    - ▹ Both:

pool1 *and* pool2.

- And:

(0)((0)(; pool1; ...;), (0)(; pool2; ...;)) for k27;

is

- Equivalent
  - To:

(0)((; pool1; ...;), (; pool2; ...;)) for k27;

- And:

do[...]; *and* do![...];

can

- Only
  - Be
    - ▹ Written:
      - In classes

in

- Which
  - These:

- ▹ Tagging
    - Statements

are

- Written.
    - ○ And:

```
class  ClassSixteen{
    protected   static   [ClassThree]   pool1;
    protected   static   [int]          k5;
    static      (...)         for       pool1;
    static      (...)         for       k5;
    public  ClassSixteen(){...}
    ⋮
}
```

is

- Equivalent
    - ○ To.

```
class  ClassSixteen{

    protected    static    [ClassThree]    pool1;

    protected    static    [int]           k5;

    (. . . )               for             pool1;

    (. . . )               for             k5;

    public  ClassSixteen(){. . . }

    ⋮

}
```

- But:

```
class  ClassSeventeen{

    protected      [ClassThree]       pool1;

    protected      [int]              k5;

    static         (. . . )     for   pool1;

    static         (. . . )     for   k5;

    publlic  ClassSeventeen(){. . . }

    ⋮

}
```

will

- Not
    - Compile.
        - ▹ And:

final (. . .) for <list>; *and* final <list> for (. . .);

are

- Equivalent
    - To:

(. . .) for <list>; *and* <list> for (. . .);

- Or
    - These:
        - ▹ Tagging
            - Statements

will

- Not
    - Be:
        - ▹ Inherited.
- And
    - So
        - ▹ They:
            - Will

be

- Ignored
    - If
        - ▹ Written:

- In interfaces,

- And
  - Also
    - ▹ Cannot
      - Be:

public *or* protected *or* private.

- And
  - We
    - ▹ Can
      - Write:

```
for(int int t in k)
    for(50 .. 100){
        ⋮
    }
```

- And:

$$\text{for(int i in } 50 \text{ .. } 10)\{\ldots\}$$

will

- Not
  - Halt,
    - ▹ Since:
      - It

will

- Not
    - Check:
        - Whether

the

- Ending
    - Integer
        - Is:
            - Greater

than

- The
    - Beginning
        - Integer.
            - But:

$$\text{for(int i in [50 .. 10])}\{\ldots\}$$

will

- Halt,
    - And
        - In:

$$\text{if (i in [100 .. 10])}\{\ldots\}$$

the

- Expression:

$$\text{i in [100 .. 10]}$$

will

- Be:

$$\text{false,}$$

- Since:

$$\text{([100 .. 10]).length} == 0.$$

- And
    - We
        - ▷ Do not
            - Allow:

$$\text{if (i in 10 .. 100)}\{\ldots\}$$

- And
    - Range
        - ▷ Operators
            - Like:

$$[\ ..\ 10,\ 50\ ..\ 100,\ 200\ ..\ ].$$

- And
    - We
        - ▷ Say
            - That:

$$\text{k5} = 8, \quad 9; \qquad \textit{and} \qquad \text{k5} = (8, \quad 9);$$

are

- Equivalent.
    - Let:

$$\text{void voidReturner([int], [int]);}$$

be

- Some
  - Method.
    - ▷ Then:
      - We

can

- Write:

$$\text{voidReturner}((8 \quad 9), \quad (10, \quad 11));$$

- And
  - Avoid
    - ▷ Methods
      - Like.

$$\text{void} \;\; \text{voidReturner}(\text{int}\ldots);$$

1.2 Trees

Let:

$$\text{i}$$

be

- Of
  - Type:

$$\text{int}.$$

- Then
  - If
    - ▷ We:
      - Attach

a

- List,
  - Say:

$$\text{someList}$$

to

- It,
  - We see that:

$$\text{someList}[0], \qquad \text{someList}[1], \qquad \ldots$$

can

- Store:

$$\textit{the first-node}, \qquad \textit{the second-node}, \qquad \ldots$$

of

- A tree.
  - And
    - ▹ So
      - If:

$$(\text{enum})\text{int} \quad z;$$

- Then:

$$z$$

will

- Not
  - Only
    - ▹ Have:

      - All things

of

- A variable
    - Of
        - Type:

int,

- But
    - A list
        - Of
            - Type:

[(enum)int]

will

- Also
    - Be
        - Attached:
            - To it.
- And
    - So
        - In:

```
int                  i      =    1000;
(enum)int            z;
z[0]                        =    10;
z[0].enum                   =    10,   20,   i;
z[0].enum[1].enum           =    100,   200,   300;
(enum)SuperClass     scz;
scz[0]                      =    new  SubClass();
scz[0].enum                 +=   new  SuperClass();
if  (z[0].enum[1].length  !=  0){...}
```

- We see that:

z[0].enum

will

- Store
  - The
    - Children:
      - Of

the

- Root,
  - And:

z[0].enum[1].enum

the

- Children
    - Of:

$$20,$$

- And
    - So:
        - Forth.
- And
    - To
        - Avoid:
            - Complications,

we

- Say
    - That:

(enum)int   int   t;

is

- Equivalent
    - To.

(enum)(int)   int   t;

- We
    - Do
        - Not
            - Allow:

(enum)(int   int)   t;

- Since:

[int (enum)(int int) int] k;

will

- Complicate.
    - But
        - ▹ We
            - Can write:

(enum){ int int t; } acz;

- And
    - We
        - ▹ Do:
            - Not

allow

- Trees
    - Of:

lists and arrays.

- But
    - We
        - ▹ Can
            - Have:

lists and arrays

of

- Trees.

- ◦ And
  - ▹ So:
    - – We

can

- Write:

$$[(\text{enum})\text{int} \quad \text{int}] \quad \text{k26};$$

$$(\text{enum})\text{int} \quad [\,][\,] \quad \text{arr} \quad = \quad \text{new} \quad (\text{enum})\text{int}[10][10];$$

- But
  - ◦ Not:

$$(\text{enum})[\text{int} \quad \text{int}] \quad \text{k}; \qquad \textit{and} \qquad (\text{enum})(\text{int} \quad [\,][\,]) \quad \text{arr};$$

- And
  - ◦ If:

$$(\text{enum})\text{int} \quad \text{z};$$

- Then:

$$\text{z.length} \quad == \quad 0.$$

- Let:

$$\text{z}, \qquad \text{z1} \qquad \textit{and} \qquad \text{z2}$$

be

- Instances
  - ◦ Of:

$$(\text{enum})\text{int},$$

- And
    - Let:

$$\mathrm{k5}$$

be

- An
    - Instance
        - Of:

$$[\mathrm{int}].$$

- Then
    - If:

$$\mathrm{z}[0,\ 0],$$

we

- Will
    - Be
        - Refering
            - To:

$$\mathrm{z}[0].\mathrm{enum}[0],$$

- And
    - If:

$$\mathrm{z}[0,\ 0,\ 1],$$

we

- Will
    - Be

        ▷ Refering
            – To:

$$\mathrm{z[0].enum[0].enum[1]},$$

- And
    - If:

$$\mathrm{z}[0,\ \ \mathrm{i}_1,\ \ \mathrm{i}_2,\ \ \ldots,\ \ \mathrm{i}_n],$$

we

- Will
    - Be
        ▷ Refering
            – To:

$$\mathrm{z[0].enum[i_1].enum[i_2].}\ \ \ldots\ \ \mathrm{.enum[i}_n],$$

- And
    - If:

$$\mathrm{z[k5]},$$

we

- Will
    - Be
        ▷ Refering
            – To:

$$\mathrm{z[k5[0]].enum[k5[1]].enum[k5[2]].}\ \ \ldots,$$

- And
    - If:

$$z[k5,\ \ k5,\ \ 0,\ \ 0],$$

we

- Will
  - Be
    - ▹ Refering
      - To:

$$z[k6],$$

- Where:

$$[int]\quad k6\quad =\quad k5,\quad k5,\quad 0,\quad 0;$$

- And
  - If:

$$z1\quad =\quad z2;$$

the

- Address
  - Of:

$$z2$$

will

- Be
  - Given
    - ▹ To:

$$z1.$$

- And
  - If:

$$z1 = z2[\,];$$

- Then:

$$z1$$

will

- Point
  - To
    - ▹ A copy
      - Of:

$$z2,$$

- And
  - If:

$$z1 = z2[k5,\ k5,\ 0,\ 0];$$

the

- Address
  - Of
    - ▹ The:
      - Leaf

located

- By:

$$[k5,\ k5,\ 0,\ 0]$$

will

- Be
  - Given

        - ▹ To:

$$\mathrm{z1}.$$

- And
    - If:

$$\mathrm{z1} \quad = \quad \mathrm{z2}[\mathrm{k5},\ \ \mathrm{k5},\ \ 0,\ \ 0\ \ ..\ \ ];$$

- Then:

$$\mathrm{z1}$$

will

- Point
    - To
        - ▹ A copy:
            - Of

the

- Subtree
    - Located
        - ▹ By:

$$[\mathrm{k5},\ \ \mathrm{k5},\ \ 0,\ \ 0].$$

- And
    - So
        - ▹ We see that:

$$[\,], \qquad [0,\ \ 0\ \ ..\ \ ] \qquad \textit{and} \qquad [\mathrm{k5},\ \ \mathrm{k5},\ \ 0,\ \ 0\ \ ..\ \ ]$$

are

- Range

- ○ Operators,
  - ▷ And:

$$[0,\ 0], \qquad [k5] \qquad \textit{and} \qquad [k5,\ k5,\ 0,\ 0]$$

are

- Location
  - ○ Operators,
    - ▷ And:
      - We

do

- Not
  - ○ Allow
    - ▷ Statements
      - Like:

$$\begin{aligned} z1[\,] &= z2[\,]; \\ z1[\,] &+= z2[\,]; \\ z1 &+= z2[\,]; \end{aligned}$$

- But
  - ○ We
    - ▷ Can
      - Write:

```
z1[ ]     =     i;

z1[k5]    +=    z2[k5,  0,  0];

z1[ ]     +=    z2[k5];

z[ ]++                 :     (. . . );

z[ ]      =     i      :     (. . . );

z[ ]      +=    i      :     (. . . );

i         =     (z     :     (. . . )).length;

i         =     (z     :     (. . . )).max;

i         =     (z     :     (. . . )).min;

i         =     (z     :     (. . . )).sum;

i         =     z.length  +  z.max  +  z.min  +  z.sum;
```

- And:

$$z(0,\ \ 0)\ \ ==\ \ \text{true},$$

- If:

$$z[0,\ \ 0]$$

is

- Defined.
  - And
    - ▷ Similarly,
      - For:

$$\text{z(k5)} \qquad \textit{and} \qquad \text{z(k5, k5)}.$$

- And
  - If:

boolean z(int...);

is

- Some
  - Method,
    - ▹ We
      - Write:

$$\text{if } (((\text{enum})\text{int})\text{z}(0))\{\ldots\}$$

- And
  - If:

$$\text{z}(0,\ \ldots,\ \text{i}_n) == \text{true} \qquad \textit{and} \qquad \text{z}(0,\ \ldots,\ \text{i}_n+1) == \text{false},$$

- Then:

$$\text{z}[0,\ \ldots,\ \text{i}_n+1]$$

will

- Be
  - Equivalent
    - ▹ To:

$$\text{z}[0,\ \ldots,\ \text{i}_n].$$

- And
  - If:

$$\text{z}[0,\ \ldots,\ \text{i}_n]\text{.enum.length} == 0,$$

- Then:

$$\text{z}[0,\ \ldots,\ \text{i}_n,\ \text{<some-int>}]$$

will

- Be
  - Equivalent
    - To:

$$\text{z}[0,\ \ldots,\ \text{i}_n].$$

- And:

$$\text{z}[-1], \qquad \text{z}[8,\ -1] \qquad \textit{and} \qquad \text{z}[\text{k5}[\,]]$$

are

- Equivalent
  - To:

$$\text{z}[0], \qquad \text{z}[0,\ 0] \qquad \textit{and} \qquad \text{z}[\text{k5}]$$

respectively.

- And
  - If:

$$\text{k5.length} \ == \ 0,$$

- Then:

$$\text{z}[\text{k5}] \ == \ \text{z}[0].$$

- Let:

$$\text{z.length} \ == \ 0,$$

- And

        - We
            - Execute:

$$\mathrm{i} \quad = \quad \mathrm{z}[0];$$

- Then:

“*default-value*”

will

- Be
    - Used
        - Instead
            - Of:

$$\mathrm{z}[0],$$

- And
    - If:

$$\mathrm{z}[0,\ \ 1] \quad = \quad 8;$$

- Then:

$$\mathrm{z}$$

will

- Be
    - Expanded
        - To
            - Length:

$$1,$$

- And:

$$z[0]$$

will

- Be
  - Used
    - ▹ Instead
      - Of:

$$z[0,\ \ 1].$$

- And
  - Define:

in *and* !in,

- And:

the for-all-statement *and* *the there-exists-statement,*

- And
  - All:
    - ▹ Boolean
      - Operations

like

- That
  - Which
    - ▹ We:
      - Did

for

- Lists.
  - And

      ▷ We
         – Can write:

$$\text{k5} \quad = \quad \text{z1} \;\; \&\& \;\; \text{z2};$$

- But
    - Not:

$$\text{z} \quad = \quad \text{z1} \;\; \&\& \;\; \text{z2};$$

- And
    - Similarly,
      ▷ For:
         – Other operations.

- Let:

$$\text{boolean} \qquad\qquad \text{boolReturner(int);}$$

$$\text{boolean} \qquad \text{boolReturner((enum)int);}$$

be

- Methods.
    - The
      ▷ Interpretation
         – Of:

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad \text{((boolean|int)boolReturner(z[ ]));}$$

- Is:

“*select all elements of*: z *that satisfy*: boolReturner,”

- And
    - That

▹ Of:

$$k5 \quad = \quad z[\,] \quad : \quad (z[\,].\text{enum.length} \;==\; 2);$$

- Is:

  "*select all elements of*: z *with exactly two children*,"

- And

  ◦ That

    ▹ Of:

$$k5 \quad = \quad z[\,] \quad : \quad ((-)z[\,] \;==\; 100);$$

- Is:

  "*select all elements of*: z *whose parent is*: 100,"

- And

  ◦ That

    ▹ Of:

$$k5 \quad = \quad z[\,] \quad : \quad (([2]-)z[\,] \;==\; 100);$$

- Is:

  "*select all elements of*: z *whose grand parent is*: 100,"

- And

  ◦ That

    ▹ Of:

$$k5 \quad = \quad z[\,] \quad : \quad ((+)z[\,] \;>\; 10); \tag{43}$$

- Is:

  "*select all elements of*: z *with all children greater than*: 10."

- Note that,
    - When
        - ▹ We execute:
            - Statement 43,

all

- Nodes
    - With
        - ▹ No:
            - Children

will

- Also
    - Be:
        - ▹ Selected.
- The
    - Interpretation
        - ▹ Of:

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad (([2]+)\text{z[ ]} \ > \ 10);$$

- Is:

"*select all elements of*: z *with all grand children greater than*: 10,"

- And
    - That
        - ▹ Of:

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad (([8+])\text{z[ ]} \ > \ 10);$$

- Is:

"*select all elements of*:   z   *with the*:   8th   *child greater than*:   10,"

- And
  - That
    - ▷ Of:

      $$\text{k5} \quad = \quad \text{z}[\,] \quad : \quad (([8+]>)\text{z}[\,] \; > \; 10);$$

- Is:

“*select all elements of*: z

with the: 8th *child’s immediate younger sibling*

greater than: 10,”

- And
  - That
    - ▷ Of:

k5 = z[ ] : ((boolean|int)boolReturner((+)z[ ]));

- Is:

“*select all elements of*: z,

such that, all its children satisfy: boolReturner,”

- And
  - That
    - ▷ Of:

k5 = z[ ] : ((boolean|int)boolReturner(([2]+)z[ ]) (1)== true);

- Is:

“*select all elements of*: z,

such that, at least one of its grand children satisfy: boolReturner.”

- And
  - Since:

z[ ]([8]>)

is

- Like:

$$z(0),$$

the

- Interpretation
  - Of:

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad (\text{z[ ]([8]>)} \quad \&\& \quad \text{!z[ ]([9]>)});$$

- Is:

“*select all elements of*: z *with exactly*: 8 *younger siblings*.”

- And:

$$\text{int} \quad \text{i} \quad = \quad -1;$$

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad ((\text{[i]+})\text{z[ ]} \; > \; 10);$$

is

- Equivalent
  - To:

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad ((\text{[0]+})\text{z[ ]} \; > \; 10); \tag{44}$$

- And
  - Statement 44
    - ▹ To:

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad (\text{z[ ]} \; > \; 10);$$

- And:

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad ((\text{[1]+})\text{z[ ]} \; > \; 10);$$

- To:

$$\text{k5} \quad = \quad \text{z}[\,] \quad : \quad ((+)\text{z}[\,] \; > \; 10);$$

- And
  - If
    - ▹ We
      - Use:

$$([8]{-}[8]{>}[8]{+}),$$

we

- First
  - Move:

“8 *steps upwards*,”

- And
  - Then:

“8 *steps to the right*,”

- And
  - Then:

“8 *steps downwards*,”

- And
  - Similarly,
    - ▹ For:

$$([8 \;\; .. \;\; 10]{-}[8 \;\; .. \;\; 10]{>}[8 \;\; .. \;\; 10]{+}),$$

- And
  - All
    - ▹ Other:

            - Combinations,

- And
    - The
        - ▹ Result:
            - Will

always

- Be:

$$\text{false,}$$

if

- The
    - Specified
        - ▹ Location:
            - Is undefined.
- And
    - So:

$$(-)\text{z}[0] \quad == \quad \text{<some-int>}$$

will

- Always
    - Be:

$$\text{false.}$$

- The
    - Interpretation
        - ▹ Of:

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad (([8+]{>}{*})\text{z[ ]} \ > \ 10);$$

- Is:

"*select all elements of*: z

with the: 8^{th} *child's all younger siblings*

are greater than: 10,"

- And
  - That
    - ▹ Of:

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad ((-{>}{*})\text{z[ ]} \ > \ 100);$$

- Is:

"*select all elements of*: z

whose parent's all younger siblings are all greater than: 100,"

- And
  - That
    - ▹ Of:

$$\text{k5} \quad = \quad \text{z[ ]} \quad : \quad ((- > *+)\text{z[ ]} \ > \ 10); \qquad (45)$$

- Is:

"*select all elements of*: z

whose parent's all younger sibling's all children are greater than: 10."

- Note that:

$$*$$

will

- Be
    - Applied
        - To:
            - The symbol

immediately

- Before
    - It.
        - And:
            - So

when

- We
    - Execute
        - Statement 45:
            - Search

will

- Be
    - Done
        - In:
            - The parent’s

all

- Younger
    - Sibling’s:
        - Families.
- And so
    - The

- ▹ Interpretation
  - Of:

$$\mathrm{k5} \quad = \quad \mathrm{z[\,]} \quad : \quad ((->*+)\mathrm{z[\,]}\ \ (1)>\ \ 10);$$

- Is:

“*select all elements of*: z

whose parent’s all younger sibling has at least one child greater than: 10,”

- And
  - That
    - ▹ Of:

$$\mathrm{k5} \quad = \quad \mathrm{z[\,]} \quad : \quad ((\mathrm{boolean}|(\mathrm{enum})\mathrm{int})\mathrm{boolReturner}((->*)\mathrm{z[\,]}));$$

- Is:

“*select all elements of*: z

whose parent all younger siblings satisfy: boolReturner.”

- And:

$$\mathrm{k5} \quad = \quad \mathrm{z[\,]} \quad : \quad (((8)>)\mathrm{z[\,]}\ \ ==\ \ 10);$$

is

- Equivalent
  - To.

$$\mathrm{k5} \quad = \quad \mathrm{z[\,]} \quad : \quad (([\ \ ..\ \ 8]>)\mathrm{z[\,]}\ \ ==\ \ 10);$$

- Let:

```
class SomeClass{
    public int i;
    public SomeClass(){}
    public int intReturnerTwo(int i){...}
}
```

- And
  - Let:

$$\text{scz}$$

be

- An
  - Instance
    - Of:

$$\text{(enum)SomeClass.}$$

- Then
  - We
    - Can
      - Write.

$$\text{[SomeClass]} \quad \text{scl} \quad = \quad \text{scz[ ]} \quad : \quad ((-)\text{scz[ ].intReturner}(8) \;\; == \;\; 80);$$

- Let:

$$\text{int} \qquad \text{intReturner(int);}$$

$$\text{int} \quad \text{intReturnerTwo(int,} \;\; \text{int);}$$

be

- Methods.
    - The
        - Interpretation
            - Of:

$$\text{z1} \quad = \quad \text{intReturner(z2[ ]);}$$

- Is:

“*transform*: z2 *to*: z1 *through*: intReturner,”

- And
    - That
        - Of:

$$\text{z1} \quad = \quad \text{intReturner}((-)\text{z2[ ]);}$$

- Is:

“*transform the mirror image of*: z2

to: z1

through: intReturner,”

- And
    - We
        - Can:
            - Say

that,

- If:

$$\text{z} \quad = \quad \text{intReturner(z1[ ], z2[ ]);}$$

- Then:

$$z[0, \ \ldots, \ m] \ == \ \text{intReturner}(z1[0, \ \ldots, \ m], \ z2[0, \ \ldots, \ m]),$$

- If:

$$z1(0, \ \ldots, \ m) \ == \ \text{true} \qquad \textit{and} \qquad z2(0, \ \ldots, \ m) \ == \ \text{true},$$

- And:

$$z(0, \ \ldots, \ m) \ == \ \text{false}$$

for

- All
  - Other:
    - Cases.
- But
  - Since
    - It:
      - Will complicate,

we

- Say
  - That,
    - If:
      - Exactly

one

- Range
  - Operator
    - Has:
      - Been juxtaposed

to

- Exactly
    - One:
        - ▷ Tree,
- And
    - That
        - ▷ Has:
            - Not

been

- Applied
    - Onto:
        - ▷ Itself,
- And
    - If
        - ▷ No:
            - Condition-part

is

- Applicable,
    - Then
        - ▷ The:
            - Tree-structure

will

- Be
    - Preserved,
        - ▷ Else:

            - All trees

to

- Which
    - A range
        - ▹ Operator:
            - Has

been

- Juxtaposed
    - Will
        - ▹ Be:
            - Seared

to

- Lists.
    - And
        - ▹ So:
            - We

do

- Not
    - Allow
        - ▹ Statements
            - Like:

$$
\begin{aligned}
z &= \text{intReturnerTwo}(z1[\,],\quad z2[\,]);\\
z &= z1[\,] + z2[\,];\\
z &= \text{intReturner}(z2[\,]) \; : \; (\ldots);
\end{aligned}
$$

- But
  - We
    - Can
      - Write:

$$
\begin{aligned}
k5 &= \text{intReturnerTwo}(z1[\,],\ \ k5[\,]); \\
k5 &= \text{intReturnerTwo}(z1[\,],\ \ z2[\,]); \\
k5 &= z1[\,],\quad z2[\,] && : \quad (\ldots); \\
k5 &= \text{intReturner}(z1[\,]) && : \quad (\ldots); \\
z &= \text{intReturner}(z1[\,]);
\end{aligned}
$$

- And
  - In
    - The
      - Expression:

$$\text{scz}[\,].\text{intReturner}(\text{scz}[\,].\text{i})$$

both

- Instances
  - Of:

$$\text{scz}[\,]$$

will

- Be seared
  - To
    - Lists,

– Since:

$$\text{scz}[\,]$$

is

- Applied
  - Onto
    - ▹ Itself.
      - Let:

$$\text{arr}$$

be

- An
  - Instance
    - ▹ Of:

$$\text{int}\ [\,].$$

- Then:

$$\text{z}[\,]$$

in

- Both:

$$\text{intReturnerTwo}(\text{z}[\,],\ \ \text{k5}[\,]) \qquad \textit{and} \qquad \text{intReturnerTwo}(\text{z}[\,],\ \ \text{arr}[\,])$$

will

- Be
  - Seared
    - ▹ To:
      - Lists,

since

- It
  - Is
    - ▹ Used
      - Along with:

a list *and* *an array*

respectively.

- And
  - So
    - ▹ We:
      - Do

not

- Allow
  - Statements
    - ▹ Like.

$$z = \text{intReturnerTwo}(z[\,],\ \ k5[\,]);$$

$$z = \text{intReturnerTwo}(z[\,],\ \ arr[\,]);$$

$$z = \text{scz}[\,].\text{intReturner}(\text{scz}[\,].i);$$

- The
  - Interpretation
    - ▹ Of:

$$z = z1[\,] * z1[\,] + \text{intReturner}(z1[\,]) * 8;$$

- Is:

  for(int ze in z1)

  The corresponding element of: z

  is: ze $*$ ze $+$ intReturner(ze) $*$ 8.

- But
  - ◦ We
    - ▹ Do:
      - Not

allow

- Statements
  - ◦ Like.

$$z = z1[\,] + z2[\,];$$

- The
  - ◦ Interpretation
    - ▹ Of:

$$[\text{int}]\ [\,]\ \text{arrA} = z1[\,] == i;$$

- Is:

  “*find the indices of*: i *in*: z1,”

- And
  - ◦ That:
    - ▹ Of:

$$[\text{int}]\ [\,]\quad \text{arrA} \quad = \quad \text{z2} \ < \ \text{z1};$$

- Is:

“*find the indices of*: z2 *in*: z1.”

- And
  - If:

$$[\text{int}]\ [\,][\,]\quad \text{arrB} \quad = \quad \text{z1}[\,] \ == \ 8;$$

- Then:

$$\text{z}[\text{arrB}[0][0]] \ == \ 8, \qquad \text{z}[\text{arrB}[0][1]] \ == \ 8, \qquad \ldots,$$

- And
  - Similarly,
    - ▹ For:
      - Other dimensions.
- And
  - If:

$$[\text{int}] \quad \text{k5} \quad = \quad \text{z}[\,] \ == \ \text{i};$$

- And:

$$\text{i} \ \ !\text{in} \ \ \text{z},$$

- Then:

$$\text{k5.length} \ == \ 1 \qquad \textit{and} \qquad \text{k5}[0] \ == \ -1.$$

- And
  - We
    - ▹ Can:
      - Give

a

- Description
  - For:
    - Index
      - Operation
- And:

$$\mathrm{scz}[\,]?$$

like

- That
  - Which
    - We:
      - Did

for

- Lists.
  - And
    - We:
      - Do

not

- Allow
  - Statements
    - Like:

$$z = z(z1, z2);$$

- And
  - If:

```
z = ;
```

- Then:

```
z.length == 0.
```

- The
  - Interpretation
    - ▷ Of:

```
z = 8( 10( 100, 100 ), 20( 200, 200 ) );
```

- Is:

```
z[0]                  =  8;
z[0].enum             =  10,  20;
z[0].enum[0].enum     =  100,  100;
z[0].enum[1].enum     =  200,  200;
```

- And
  - That
    - ▷ Of:

```
z = 8( z1, 10 );
```

- Is.

```
z[0]           =  8;
z[0].enum      =  <some-int>,  10;
z[0].enum[0]   =  z1;
```

- And
    - Similarly,
        - For:

$$z \quad = \quad 8(\;\; z1[\,], \;\; 10 \;\;);$$

- And:

$$\begin{aligned} z &= 8(\;\; k5, \;\; 100 \;\;); \\ k5 &= z[0].enum; \\ z[0].enum &= k5; \end{aligned}$$

is

- Equivalent
    - To:

$$\begin{aligned} z &= 8(\;\; k5[\,], \;\; 100 \;\;); \\ k5 &= z[0].enum[\,]; \\ z[0].enum &= k5[\,]; \end{aligned}$$

- But
    - We
        - Do:
            - Not

allow

- Statements

    - ◦ Like.

$$z \quad = \quad 8(\ z1(\ 100\ )\ );$$

$$z \quad = \quad 8(\ k5(\ 100\ )\ );$$

- Let:

$$\text{int}\ \ \text{i(int)}; \qquad (46)$$

be

- Some
    - ◦ Method,
        - ▹ And
            - – Let:

$$\text{i}$$

be

- Of
    - ◦ Type:

$$\text{int.}$$

- Then:

$$z \quad = \quad i(10);$$

will

- Be
    - ◦ Equivalent
        - ▹ To:

```
int   j   =   i;
z         =   j(10);
```

- And:

```
z   =   (int)i(10);
```

- To:

```
int   j   =   i(10);
z         =   j;
```

- And:

```
(enum)int  [ ]   arr     =     (   1(  10  ),    2(  20  )    );
```

should

- Be
  - Rewritten
    - As:

```
(enum)int        z1    =    1(  10  ),    z2    =    2(  20  );
(enum)int  [ ]   arr   =    (    z1,    z2    );
//  And  similarly,  for  lists.
```

- And:

```
if  (z1  ==  8(  10  )){...}
```

- As.

  $(\text{enum})\text{int} \quad z2 \quad = \quad 8(\;\; 10 \;\;);$

  $\text{if} \;\; (z1 \;\; == \;\; z2)\{\dots\}$

- The
  - Interpretation
    - Of:

$$[\text{int}] \quad k5 \quad = \quad \text{z.depth};$$

- Is:

"*find the length of all paths from the root to all leaves with no children in*: z."

- And
  - If
    - We
      - Use:

$$\text{enum.subtrees}[2][8][!(([0] \;\; == \;\; 1), \;\; ((+)[0] \;\; > \;\; [0]), \;\; ([0] \;\; == \;\; 8))],$$

the

- Interpretation
  - Will
    - Be:

"*select all binary subtrees of depth*: 8 *with no path defined by*: $(([0] \;\; == \;\; 1), \;\; ((+)[0] \;\; > \;\; [0]), \;\; ([0] \;\; == \;\; 8))$."

- And

        ◦ If:

$$\text{z.transient} \quad = \quad (\text{t})\{\ldots\};$$

- Then:

$$\text{z.state}$$

will

- Be
    - An
        - ▹ Instance
            - Of:

$$[\text{int}],$$

- And
    - If:

$$\text{z}[\text{i}+];$$

then

- After
    - Executing
        - ▹ The
            - Transient-block:

$$\text{z.state} \quad += \quad \text{i};$$

will

- Be
    - Executed.
        - ▹ The
            - Interpretation of:

$$z>;$$

is

- Similar,
    - Except
        - That:
            - After executing

that

- Block:

$$z.state[z.state.length - 1]++;$$

will

- Be executed.
    - And
        - Similarly,
            - Define:

$$z<; \qquad \textit{and} \qquad z;$$

- And
    - If:

$$z-;$$

then

- After
    - Executing
        - That
            - Block:

$$z.state = z.state[\ ..\ z.state.length - 1];$$

will

- Be
    - Executed.
        - ▹ And
            - If:

$$z{>}*;$$

first

- That
    - Block
        - ▹ Will be
            - Executed for:

$$z[z.state],$$

- And
    - Then
        - ▹ For:
            - All

its

- Elder-siblings.
    - And
        - ▹ Similarly,
            - Define:

$$z{<}*;$$

- And
    - If:

$$z+*;$$

first

- That
    - Block
        - Will be
            - Executed for:

$$z[z.state],$$

- And
    - Then
        - For:
            - Its

first

- Child,
    - The
        - Second:
            - Child,
- And
    - So:
        - Forth.
- And
    - If:

$$z+**;$$

first

- That

- ○ Block
  - ▹ Will be
    - – Executed for:

z[z.state],

- And
  - ○ Then
    - ▹ For:
      - – Its

first

- Child,
  - ○ The
    - ▹ First:
      - – Child's

first

- Child,
  - ○ And
    - ▹ So:
      - – Forth,
- Then
  - ○ Its
    - ▹ Second:
      - – Child,
- Its
  - ○ Second
    - ▹ Child's:
      - – First-child,

- And
  - So:
    - ▹ Forth.
- And
  - If:

$$\text{z}-*;$$

first

- That
  - Block
    - ▹ Will be
      - Executed for:

$$\text{z[z.state]},$$

- And
  - Then
    - ▹ For
      - Its:

parent, *grandparent*, ...

- And
  - By
    - ▹ Default:

$$\text{<tree-name>.state.length} \quad == \quad 0.$$

- And
  - We
    - ▹ Can:

- Give

a

- Description
    - Like
        - That:
            - Which

we

- Did
    - For:
        - Lists.
- And
    - We
        - Can
            - Write:

$$k5 = (0)z;$$

to

- Get
    - The
        - List:
            - Of

all

- Different
    - Elements
        - In:

z.

- But
  - We
    - ▹ Do not
      - Allow:

$$z1 = (0)z2;$$

- And:

(enum)(null)int i; *and* (null)(enum)int i;

are

- Equivalent.
  - And
    - ▹ Similarly:
      - For

all

- Combinations
  - Of:

(+), (null) *and* (enum).

- And
  - If:

partial (enum)SomeClass z3;

- Then:

z3

can

- Only
    - Hold:
        - Instances

of

- Subclasses
    - Of:

SomeClass,

- And
    - Similarly,
        - For:

lists *and* *arrays.*

- Let:

k3

be

- An
    - Instance
        - Of:

[int int int].

- Then
    - We
        - Can
            - Write:

```
(enum)int [ ] arr;

arr = (enum.subsets[ ][(([0] < i, [1] == (−)[0])])k3;
```

for

- Hierarchical
  - Queries,
    - ▷ And:

```
[int] k5 = ([8]−)k3[ ][0] > k3[ ][1] && k3[ ][1] < ([8]+)k3[ ][2];
```

- And:

```
int [ ][ ][ ] arrQ = . . . ;

k3                 = arrQ[ ][ ][ ] == ([8]!+!)arrQ[ ][ ][ ];
```

- And
  - Similarly,
    - ▷ For:
      - – Other dimensions.
- The
  - Interpretation
    - ▷ Of:

```
[int int] k = . . . ;

do{

    Perform an operation on: k.

}transient(k)
```

- Is:

"*perform an operation on*: k *until no more changes can be made in it*."

- And
  - If:

```
[int int] k1 = ..., k2 = ...;
do{
    Perform some operations on: k1 and k2.
}transient(k1, k2)
```

the

- Construct
  - Will
    - ▹ Continue:
      - Looping

until

- No
  - More
    - ▹ Changes:
      - Can

be

- Made
  - In
    - ▹ Both:

k1 *and* k2.

- And
  - Similarly,
    - For:

```
int      [ ]      arr   =   ...;
(enum)int         z     =   ...;
[int  int]        k1    =   ...,   k2   =   ...;
int               i     =   ...;
do{
    ⋮
}transient(k1,  i,  k2,  arr,  z)
```

- And
  - When
    - We
      - Say:

a tree *or* *a list* *or* *an array*

has

- Not
  - Changed,
    - We:
      - Mean

that,

- Its
    - Size
        - ▹ Has:
            - – Not changed,
- Or:

values *or* *addresses*

stored

- In
    - Them
        - ▹ Has:
            - – Not changed,
- Or
    - The
        - ▹ Order:
            - – In them

has

- Not
    - Changed.
        - ▹ And:
            - – So

if

- An
    - Object,
        - ▹ Say:

obj

is

- Stored
    - In:

a tree *or* *a list* *or* *an array,*

- And
    - There
        - ▹ Was:
            - A change

in

- Its
    - Field
        - ▹ Value:
            - We

say

- That:

obj

has

- Changed,
    - But
        - ▹ That:

tree *or* *list* *or* *array*

did

- Not
  - Change.
    - ▹ And
      - So:

```
ClassOne co = ...;
do{
    ⋮
}transient(co)
```

will

- Continue
  - To
    - ▹ Loop:
      - As

long

- As
  - There
    - ▹ Is:
      - A change

in

- The
  - Address
    - ▹ Stored
      - At:

co.

- But:

```
int int t = ...;
do{
    ⋮
}transient(t)
```

is

- Equivalent
    - To:

```
int int t = ...;
do{
    ⋮
}transient(t[0], t[1])
```

- And
    - It
        - Will:
            - Continue looping

as

- Long

- As
    - There:
        - Is

a

- Change
    - In:

$t[0]$ *or* $t[1]$.

- And
    - If:

```
class ClassTwo{
    public [int int] k;
    public ClassTwo(){}
    public void inc(){ k += 10 10; }
}
```

- Then:

```
ClassTwo ct = ...;
do{
    ct.inc();
}transient(ct.k)
```

will

- Loop
    - Forever,
        - ▹ Since:
            - – The program

will

- Only
    - Acknowledge
        - ▹ Changes:
            - – That

are

- Explicitly
    - Made
        - ▹ Inside:
            - – The construct.
- And
    - Since
        - ▹ It:
            - – Is essential

that

- An
    - If-statement
        - ▹ Be:
            - – Used

to

- Halt
    - The
        - Process:
            - Of change,

we

- Say
    - That:

```
[int  int]   k   =   10  10;
do{
    k[0][0]      =   10;
}transient(ct.k)
```

will

- Loop
    - Forever,
        - Even:
            - Though,

there

- Is
    - No
        - Real
            - Change in:

k.

- And
    - If
        - ▷ We
            - Execute:

continue;

we see that,

- Even
    - If
        - ▷ There:
            - Has

been

- No
    - Change
        - ▷ In:

the variable *or* *the tree* *or* *the list* *or* *the array*,

we

- Expect
    - Changes
        - ▷ In:
            - The future.
- And
    - So
        - ▷ If:

continue;

the

- Program
  - Will
    - ▹ Assume:
      - That,

there

- Has
  - Been:
    - ▹ A change,
- And
  - Continue
    - ▹ Looping.
      - Let:

boolean boolReturner(string);

be

- Some
  - Method,
    - ▹ And
      - Let:

string1, string2 *and* string3

be

- Instances
  - Of:

string.

- Then:

$$\text{string1[i]}, \qquad \text{string1[i ..]}, \qquad \text{string1.trim},$$

$$\text{string1.lcase}, \qquad \text{string1.ucase}, \qquad \text{string1.length}$$

are

- Well
  - Understood.
    - ▹ And:

$$\text{string1.trim?} == \text{true},$$

- If:

$$\text{string1}$$

is

- Trim,
  - And
    - ▹ Similarly,
      - For:

$$\text{string1.lcase?} \qquad \textit{and} \qquad \text{string1.ucase?}.$$

- And
  - Since:

$$\text{(!)string1}$$

is

- Like
  - Negating

        - ▹ Or:
            - Nullifying

all

- Things
    - That:
        - ▹ Can

be

- Negated
    - Or
        - ▹ Nullified
            - In:

$$\text{string1},$$

we

- Can
    - Write:

$$(!)\text{string1} == \text{string2}$$

for

- Comparison
    - Ignoring
        - ▹ Case.
            - And:

$$(\text{string1 in string2}) == \text{true},$$

- If:

$$\text{string2} \qquad \textit{contains}: \qquad \text{string1},$$

- And:

(string1 in.startswith string2) == true,

- If:

string2 *starts with*: string1,

- And:

(string1 in.endswith string2) == true,

- If:

string2 *ends with*: string1,

- And:

(string1 in.exp reg-Exp) == true,

- If:

string1 *matches*: reg-Exp.

- And:

!in.startswith, !in.endswith *and* !in.exp

are

- The
  - Negations
    - Of:

in.startswith, in.endswith *and* in.exp

respectively.

- And
  - We

- ▹ Can
  - – Write:

(!)string1 in string2 *or* string1 in (!)string2

for

- Ignoring
  - ○ Case.
    - ▹ The
      - – Interpretation of:

[int] k5 = (!)string2 in string1 || (!)string3 in string1;

- Is:

“*find the indices of*: string2 *and* string3

in: string1 *ignoring case*,”

- And
  - ○ That
    - ▹ Of:

[string] k21 = (!)(enum.substrings[ ][reg-Exp])string1; (47)

- Is:

“*select all substrings of*: string1 *that matches*: reg-Exp

ignoring case.”

- And:

[string] k21 = (enum.substrings[ ][reg-Exp])(!)string1;

- And
  - ○ Statement 47

      - ▹ Are:
        - Equivalent.

- And
  - Similarly,
    - ▹ Using:

      enum.substrings[ ][reg-Exp || boolReturner].

- And
  - We
    - ▹ Can:
      - Give

a

- Description
  - Like
    - ▹ That:
      - Which

we

- Did
  - For:
    - ▹ Lists.

- The
  - Interpretation
    - ▹ Of:

string1(string2, reg-Exp);

- Is:

“*replace all substrings in*: string1

that matches: reg-Exp

by: string2.”

- And:

string1 / reg-Exp *and* string1 / reg-Exp / i (48)

can

- Be
  - Used
    - ▹ Instead
      - Of:

string1.split(reg-Exp) *and* string1.split(reg-Exp, i)

respectively.

- And so
  - Expressions 48
    - ▹ Will:
      - Return

an

- Instance
  - Of:

[string].

- Let:

k32

be

- An
  - Instance
    - Of:

$$[\text{char}].$$

- The
  - Interpretation
    - Of:

$$\text{k32} \quad = \quad \text{string1}; \tag{49}$$

- Is:

“*for all*: i, k32[i] == string1[i].”

- And:

$$\text{string1} \quad = \quad \text{k32};$$

is

- The
  - Inverse
    - Of:
      - Statement 49
- And
  - We
    - Can
      - Write:

```
k32      =   string1,   string2;

string1  =   k32,       k32;
```

- And:

$$(-)\text{string1};$$

for

- Reversing.
  - And:

$$++ \qquad\qquad \textit{and} \qquad\qquad --$$

can

- Be
  - Used
    - To:
      - Get

the

- Next
  - And
    - Previous:
      - Strings

in

- The
  - Lexicographical
    - Order.
      - Note that:

$$\text{string1}++; \qquad\qquad \textit{and} \qquad\qquad \text{string1}[\,]++;$$

are

- Different.

- ◦ And:

  $((-)$`"a"` $+$ $(-)$`""` $+$ $--$`""` $+$ `"b"`$)$ $==$ `"ab"`.

- And
  - ◦ If:

string1 $*=$ abc $*$ abc $\backslash$ abc $\backslash$`"` abc $//$ abc $=$ abc 8;

- Then:

string1 $==$ `"`abc $*$ abc $\backslash\backslash$ abc $\backslash\backslash\backslash$`"` abc $//$ abc $=$ abc 8`"`.

- Or
  - ◦ The
    - ▹ Value
      - – Of:

string1

will

- Be
  - ◦ The
    - ▹ Trim:
      - – Of

the

- Exact
  - ◦ Sequence
    - ▹ Of:
      - – Characters

between

- The
  - First:

$$‘*=’$$

- And
  - The
    - First:

$$‘;’.$$

- And
  - So
    - If:

$$\begin{aligned} \text{string1} \quad *= \quad & \text{abc} * \text{abc}\ \text{abc} \\ & \text{abc}\ \text{abc} = 20; \end{aligned}$$

- Then:

$$\text{string1} \quad == \quad \texttt{"}\text{abc} * \text{abc}\ \text{abc}\ \text{abc}\ \text{abc} = 20\texttt{"}.$$

- And
  - If:

$$\text{int} \quad i \quad = \quad 50;$$

$$\text{string1} \quad *= \quad \text{abc} + \text{abc}\ i * j;$$

$$\text{int} \quad j \quad = \quad 100;$$

$$\text{string2} \quad *= \quad \text{abc} + \text{abc}\ i * j;$$

- Then:

$$\text{string1} \quad == \quad \texttt{"}\text{abc} + \text{abc}\ 50 * \text{j}\texttt{"},$$

- And:

$$\text{string2} \quad == \quad \texttt{"}\text{abc} + \text{abc}\ 50 * 100\texttt{"}.$$

- Or
  - When
    - We
      - Execute:

$$\text{string2} \quad *= \quad \text{abc} + \text{abc}\ \text{i} * \text{j};$$

the

- Program
  - Will
    - Replace:

$$\text{i} \qquad \textit{and} \qquad \text{j}$$

- In:

$$\texttt{"}\text{abc} + \text{abc}\ \text{i} * \text{j}\texttt{"},$$

- With:

$$50 \qquad \textit{and} \qquad 100$$

$$\textit{respectively},$$

- And
  - Generate
    - A new:
      - String,

- And
    - Then
        - ▹ Assign
            - It to:

$$\text{string2.}$$

- And
    - So:

$$\text{string1} \quad *= \quad \text{abc} \ ';';$$

should

- Be
    - Rewritten
        - ▹ As.

```
char c = ';';

string1 *= abc 'c';
```

- Let.

```
class SomeClass{

    public int i1;

    public SomeClass(){}

}
```

- And
    - We
        - ▹ Execute:

```
SomeClass   p   =   null,   p2   =   new  SomeClass();
p2.i1           =   10;
string1         *=  p  p.i1  p.i2  p2  p2.i1  p2.i2;
                // Note that: p will be replaced by:   "",
                // p.i1                   by:          "",
                // p.i2                   by:          .i2,
                // p2                     by:          "",
                // p2.i1                  by:          10,
                // And: p2.i2             by:          .i2.
```

- Then:

```
string1   ==   "   .i2   10  .i2".
```

- And
    - If:

```
string1   *=   for  while  break  abc;
```

- Then:

```
string1   ==   "for  while  break  abc".
```

- Or
    - Language
        - ▹ Keywords:
            - Will

have

- No
    - Effect
        - ▹ Between
            - The first:

$$‘*=’,$$

- And
    - The
        - ▹ First:

$$‘;’.$$

- And
    - The:
        - ▹ Right
            - Hand-side

of

- These
    - Statements
        - ▹ Cannot:
            - Be

given

- To
    - Methods
        - ▹ Like:

@void codeReturner(@native);

- And

- By
  - ▹ Default:
    - Strings

will

- Be
  - Initialized
    - ▹ To:

"".

- And
  - If:

enum e = { v0, v1, v2 };

then

- The
  - Value
    - ▹ Of:

e

can

- Only
  - Be
    - ▹ One
      - Among:

v0 *or* v1 *or* v2.

- And so
  - We

            ▷ Should:
                – Give

the

- Range
    - Of
        - ▷ Values:
            - It

can

- Store
    - When
        - ▷ It:
            - Is declared.
- And so
    - We
        - ▷ Cannot
            - Write:

```
enum    e;
```

- And
    - In:

```
enum    e   =   {   v0,   v1,   v2   };
e           =   v100;
```

the

- Last

    ◦ Line
        ▹ Will:
            – Produce

an

- Error.
    ◦ And
        ▹ The:
            – Range

of

- These
    ◦ Variables
        ▹ Cannot:
            – Be changed

after

- Declaration.
    ◦ But
        ▹ We
            – Allow:

```
enum  e1  =  {  v0  };
enum  e2  =  e1 + {  v1  };
```

- And
    ◦ In:

$$\text{int} \quad v0 \quad = \quad \ldots;$$

$$\text{enum} \quad e \quad = \quad \{ \quad v0, \quad v1 \quad \};$$

$$e \quad = \quad v0;$$

$$\text{if} \ (e \ == \ v0)\{\ldots\}$$

- When:

$$e \quad = \quad v0;$$

is

- Compiled,
    - The:
        - Compiler

can

- Understand
    - That:

$$v0$$

is

- Allowed
    - For:

$$e,$$

- And
    - So
        - Generate:
            - Code.

- And
    - If:

enum e = { v0, v1 };

- Then:

e

will

- Be
    - Initialized
        - ▷ To:

v0,

which

- Is
    - The
        - ▷ First:
            - – Value

in

- Its
    - Range.
        - ▷ And:
            - – We

do

- Not
    - Allow:

enum e = { v0, v1, v1 };

- And
    - If:

```
enum    e    =    {    v0,    v1,    v2    };
e++;
e++;
e++;
e--;
e          =    -100;
e          =    1;
e          =    100;
string1    =    (string)e  +  " "  +  (int)e;
```

the

- Sequence
    - Of
        - Values
            - In:

$$e$$

will

- Be:

v0,   v1,   v2,   v0,   v2,   v0,   v1,   v2,

- And:

$$\text{string1} == \texttt{"v2 2"}.$$

- And
  - If:

```
enum    e     =     {   v0["tag0"],   v1   };
e[e]          =     "";
string1       =     "tag1";
e[v1]         =     string1;
e++;
e++;
e             =     "abc";
string1       =     e[v0]  +  " "  +  e[v1]  +  " "  +  e[0];
```

the

- Sequence
  - Of
    - Values
      - In:

e

will

- Be:

```
v0["tag0"],    v0[""],    v1["tag1"],    v0[""],    v0["abc"],
```

- And:

string1 == "abc tag1 abc".

- And
  - Tags
    - ▹ Should:
      - Be

of

- Type:

string.

- And
  - We
    - ▹ Do not
      - Allow.

```
enum  e1  =  {...};
enum  e2  =  {...};
e1        =  e2;
```

- And
  - So
    - ▹ We:
      - Do

not

- Allow
  - Methods

        ▷ Like:

enum someMethod(enum);

- And:

trees, lists and arrays

- Of:

enum.

1.3 Database

Consider

- The
    - Class.

```
protected static class SomeStaticClass{
    ⋮
}
```

- Then
    - Since
        ▷ It:
            – Is

a

- Static
    - Class,
        ▷ All

- Its:

fields *and* *methods*

will

- Be:

static.

- And
  - If:

a static-class *extends*: *a non-static-class*,

only

- Static
  - Members
    - ▷ Will:
      - Be inherited.
- And
  - Partial:
    - ▷ Static
      - Classes

should

- Be
  - Extended.
    - ▷ And:
      - When

the

- Program:

        - Starts
            - And
                - Halts,

the

- Default
    - Constructors
        - And:
            - Finally-blocks

of

- All
    - Non
        - Partial:
            - Static-classes

will

- Be executed
    - Without
        - Any:
            - Specific-order.
- And:
    - Non
        - Static
            - Classes

cannot

- Extend:
    - Static
        - Classes,

- Or
    - Implement:
        - ▷ Static
            - Interfaces.
- And
    - If:

```
static class SomeStaticClass{
    public native int i;
    ⋮
}
```

- Then:

i

will

- Be:
    - A native
        - ▷ Field,
- And
    - Its:
        - ▷ Semantics

is

- Similar

- To
  - ▹ That:
    - – Of

non

- Native
  - Field,
    - ▹ Except:
      - – That

only

- Values
  - In
    - ▹ Such:
      - – Fields

of

- Static
  - Classes
    - ▹ Can:
      - – Be committed

into

- The
  - Database.
    - ▹ And
      - – So:

public native int i;

written

- In
    - Non
        - ▷ Static:
            - Classes

will

- Be
    - Equivalent
        - ▷ To:

```
public int i;
```

- And
    - Since
        - ▷ For every:
            - Database-table,

there

- Exists
    - An
        - ▷ Equivalent:
            - List,

we

- Can
    - Rewrite:

```
create table Table1(
    col1          int      NOT NULL,
    col2          int      CHECK      (col2 > 0),
    col3          varchar(20),
    CONSTRAINT    pkID     PRIMARY KEY(col1, col3)
)
create table Table2(
    col1          int      PRIMARY KEY,
    col2          int      UNIQUE,
    col3          varchar(20),
    FOREIGN KEY (col3)     REFERENCES Table1(col3)
                           ON DELETE CASCADE
)
```

- As:

```
public static class SchemaOne{
    public native [int int string] Table1;
    public native [int int string] Table2;
    SchemaOne{
        Table1[ ][0] != null;
        Table1[ ][1] > 0;
        (#)("pkID")Table1[ ][0, 2];
        // Or. ("pkID")(#)Table1[ ][0, 2];
        (#)Table2[ ][0];
        (0)Table2[ ][1];
        Table1[ ][2].length < 21;
        Table2[ ][2].length < 21;
        (!)Table2[ ][2] =<= Table1[ ][3];
        // All cascading will be done automatically.
    }
    ⋮
}
```

- And
  - If:

```
native{
    ⋮
}
catch(. . . ){. . . }
```

then

- all
    - Changes
        - ▹ Made:
            - – Inside

that

- Block
    - In
        - ▹ All:
            - – Native-fields

of

- All
    - Static
        - ▹ Classes:
            - – Will

be

- Committed.
    - But
        - ▹ If:

          - An exception

is

- Thrown,
    - Then
        - ▷ All:
            - Changes

will

- Be
    - Rolled-back.
        - ▷ Or
            - If:

```
public static class SchemaTwo{
    public native int i, j;
    ⋮
}
```

- And
    - If:

$$\text{void methodOne()\{ SchemaTwo.i} = \ldots; \} \tag{50}$$
$$\text{void methodTwo()\{ SchemaTwo.j} = \ldots; \} \tag{51}$$

are

- Methods,
    - And
        - ▷ We

- Execute:

```
void someMethod(){
    native{
        SchemaTwo.i = ...;
        methodTwo();
    }
    catch(...){...}
}
```

only

- The
  - ○ Value
    - ▷ In:

SchemaTwo.i

will

- Be
  - ○ Committed.
    - ▷ But
      - If:

```
void someMethodTwo(){
    native{
        SchemaTwo.i    =    ...;

        methodOne();

    }
    catch(...){...}

}
```

then

- Method 50
    - Will:
        - Throw

an

- Exception,
    - Since:

SchemaTwo.i

is

- Locked
    - By.

void someMethodTwo();

- Or
    - Inside
        - That:
            - Block,

all

- Native
    - Locations
        - ▹ In which:
            - Changes

are

- Explicitly
    - Made
        - ▹ Will be:
            - Write-locked,
- And
    - All:
        - ▹ Native
            - Locations

that

- Are
    - Explicitly
        - ▹ Read:
            - Will

be

- Read
    - Locked.
        - ▹ Let:

```
class  SomeClass{
    public          int           i;
    protected       int           i2;
    private         int           i3;
    public          SomeClass     next;
    public  SomeClass(){}
}
static  class  SchemaThree{
    public   native   SomeClass     sc;
    public   native   [int]         k5;
    public   native   [SomeClass]   scl;
}
```

- And
  - We
    - Execute:

```
SomeClass    obj       =       ...;
native{
    SchemaThree.scl    +=    obj;
}
```

- Then:

$$\text{obj}$$

will

- Be
    - Serialized
        - ▷ Or
            - Committed into:

$$\text{SchemaThree.scl.}$$

- But:

$$\begin{aligned}
&\text{native}\{\\
&\quad \text{SomeClass obj} &= \quad &\text{SchemaThree.sc;}\\
&\quad \text{obj.i} &= \quad &\ldots;\\
&\}
\end{aligned}$$

will

- Not
    - Commit:
        - ▷ Anything.
- And
    - If:

$$\text{native}\{\ldots\} \tag{52}$$

could

- Not
    - Acquire:
        - ▹ All

the

- Necessary
    - Locks,
        - ▹ The:
            - Program

will

- Not
    - Wait
        - ▹ To:
            - Acquire

those

- Locks,
    - But
        - ▹ Ignore:
            - Statement 52.
- And
    - So
        - ▹ If:

$$\text{boolean} \quad b \quad = \quad \text{native}\{\ldots\}$$

- Then:

$$b \ == \ \text{true},$$

if

- The
  - Program
    - Enters:
      - That block.
- And:

```
native{
    ⋮
    someLabel:    boolean    b    =    native{...}catch(...){...}
    ⋮
}
```

will

- Be
  - Converted
    - To:

```
native{
    ⋮
    boolean    b    =    true;
    someLabel:    try{...}catch(...){...}
```

$\vdots$

}

- And
  - If:

SchemaThree.scl $\quad += \quad \ldots;$

SchemaThree.scl.commit;

then

- All
  - Uncommitted
    - ▹ Things
      - – In:

SchemaThree.scl

will

- Be committed.
  - And
    - ▹ Similarly,
      - – For:

SchemaThree.scl.rollback;

- And
  - If:

SchemaThree.commit;

then

- All
  - Uncommitted
    - ▹ Things
      - – In:

SchemaThree

will

- Be committed.
  - And
    - ▹ Similarly,
      - – For:

SchemaThree.rollback;

- And
  - We
    - ▹ Can
      - – Write:

this.commit; *and* this.rollback; (53)

or

commit; *and* rollback; (54)

in

- Static
  - Classes.
    - ▹ And:
      - – Statements 53 and 54

written

- In:
    - Non
        - Static
            - Classes

will

- Be
    - Ignored.
        - And
            - If:

```
static.commit;
```

then

- All
    - Uncommitted
        - Things:
            - In

all

- Static
    - Classes
        - Will be:
            - Committed.
- And
    - Similarly,
        - For:

```
static.rollback;
```

- And:

commit *and* rollback

are

- Not keywords.
    - And
        - ▹ If:
            - Native-fields

have

- Been
    - Initialized
        - ▹ In:
            - The program,

then

- Those
    - Values
        - ▹ Will:
            - Be used

if

- Their
    - Database
        - ▹ Gets:
            - Deleted.
- In
    - Section 3
        - ▹ We:
            - Will

give

- Details
  - Of
    - ▷ Statements
      - Like.

$$\text{<data-type>} \quad \text{<variable-name>} \quad = \quad (\,)\{\ldots\};$$

- And so
  - We
    - ▷ Can
      - Write:

```
static class SchemaFour{

    public  native  [int int]  Table;

    private  [int]  view  =  (){ return Table[ ][0] : (...); };

    public void someMethod(){

        [int]  view      =  (){ return Table[ ][1] : (...); };

        ⋮

    }

}
```

for

- Views.
  - Let:

$$k3$$

be

- An
  - Instance
    - Of:

$$[\text{int} \quad \text{int} \quad \text{int}].$$

- Then:

$$(k3[\,]).\text{before.update} \tag{55}$$

is

- The
  - Copy:
    - Of

the

- Elements
  - In:

$$k3[\,]$$

that

- Will
  - Be
    - Updated:
      - Just before

we

- Execute:

$$\text{k3}[\,][0]{+}{+} \quad : \quad (\ldots);$$

- And:

$$(\text{k3}[\,][0, \quad 1] \quad : \quad (\text{k}[0] \quad == \quad 10)).\text{before.update} \tag{56}$$

is

- The
    - Copy:
        - Of

the

- Elements
    - In:

$$\text{k3}[\,][0, \quad 1]$$

that

- Will
    - Be:
        - Updated,
- And
    - Also
        - That
            - Satisfy:

$$(\text{k}[0] \quad == \quad 10)$$

just

- Before
    - We
        - Execute:

$$\text{k3[ ][0]++} \quad : \quad ((\text{k[0]} \;==\; 10) \;||\; (\text{k[1]} \;==\; 20));$$

- And
  - Similarly:

$$(\text{k3[ ]} \quad : \quad (\text{k[0]} \;==\; 10)).\text{after.update} \tag{57}$$

is

- The copy
  - Of
    - ▷ The:
      - New values,
- And
  - Also
    - ▷ That
      - Satisfy:

$$(\text{k[0]} \;==\; 10)$$

after

- Executing.

$$\text{k3[ ][0]++} \quad : \quad ((\text{k[0]} \;==\; 10) \;||\; (\text{k[1]} \;==\; 20));$$

- Note that,
  - Since
    - ▷ In
      - Expressions 55 56 and 57:

$$\text{k3[ ]} \qquad \textit{and} \qquad \text{k3[ ][0, 1]} \;:\; (\text{k[0]} \;==\; 10)$$

are

- Enclosed

- In
  - Between:

(*and*),

we see that,

- Expressions 55 56 and 57
  - Are:
    - Lists.
- And
  - So
    - If:

```
static  class  SchemaFive{

    public    native    [int  int]    k;

    public    native    [int  int]    k2;

    public    [int  int]              view    =    (){...};

    void  methodOne([int  int])                  for
                    (k[ ]    :    (k[ ][0]  ==  10)).before.update;

    void  methodTwo([int  int])                  for
                    (k[ ]    :    (k[ ]    :    (...)).before.insert;

    void  methodThree(int  int,  int  int)       for    view.update;

    private  void  methodOne([int  int]  k1){...}

    public  void  methodTwo([int  int]  k1){...}

    private  int  methodThree(int  int  ov,  int  int  nv){...}

    ⋮

}
```

- And we
  - Make
    - ▹ An updation
      - In:

k,

then

- Just

    - Before
        - That:
            - A copy

of

- The
    - Elements
        - That
            - Satisfy:

$$(\text{k}[\,][0] \;\; == \;\; 10)$$

will

- Be
    - Given
        - To:

$$\text{void} \;\; \text{methodOne}([\text{int} \;\; \text{int}]);$$

- And
    - If:

$$\text{k} \quad += \quad 10 \;\; 10, \quad 20 \;\; 20;$$

- Then:

$$10 \;\; 10, \quad 20 \;\; 20$$

will

- Be
    - Given
        - To:

$$\text{string} \;\; \text{methodTwo}([\text{int} \;\; \text{int}]);$$

- And
  - If:

```
void  methodOne([int  int])    for
               (k[ ]    :    (bool-Exp$_1$)).before.delete;

void  methodTwo([int  int])    for
               (k[ ]    :    (bool-Exp$_2$)).before.delete;
```

then

- The
  - Syntax
    - Tree
      - Of:

$$\text{bool-Exp}_1$$

cannot

- Be
  - A subtree
    - Of
      - That of:

$$\text{bool-Exp}_2,$$

- And
  - Vice
    - Versa.
      - And:

```
SchemaFive.view[ ][0]    +=    8    :    (bool-Exp);
```

will

- Be
    - Converted
        - To:

```
[int  int]     k   =   SchemaFive.view[ ]   :   (bool-Exp);
for(int   i   in   0 ..  k.length){
    int  int   t   =   (k[i][0]  +  8)  k[i][1];
    int        i1  =   SchemaFive.methodThree(k[i],  t);
}
```

- And:

```
        SchemaFive.view[ ][0]   =   8   :   (bool-Exp);
```

- To:

```
[int  int]     k   =   SchemaFive.view[ ]   :   (bool-Exp);
for(int   i   in   0 ..  k.length){
    int  int   t   =   8  k[i][1];
    int        i1  =   SchemaFive.methodThree(k[i],  t);
}
```

- And:

```
           SchemaFive.view[ ][0]++   :   (bool-Exp);
```

- To:

```
[int  int]      k    =    SchemaFive.view[ ]    :    (bool-Exp);
for(int    i    in    0  ..  k.length){
    int  int    t    =    (k[i][0]  +  1)  k[i][1];
    int         i1   =    SchemaFive.methodThree(k[i],  t);
}
```

- And:

```
        (int  i)SchemaFive.view[ ][1]−−    :    (bool-Exp);
```

- To:

```
int              i;
[int  int]       k    =      SchemaFive.view[ ]    :    (bool-Exp);
for(int    j    in    0  ..  k.length){
    int  int     t    =      k[j][0]  (k[j][1]  −  1);
    i                 +=     SchemaFive.methodThree(k[j],  t);
    //  If:  +=  is  undefined  for  the  receiver,
    //  then:  =  will  be  used.
}
```

- And

- Similarly,
  - For:

```
int i;

(i)SchemaFive.view[ ][1]−−;
```

- And
  - Statements
    - Like:

```
void methodOne([int int]) for (k[ ] : (. . . )).before.insert;
```

should

- Be
  - Written
    - In:
      - The class

in

- Which:

the table *or* *the view*

has

- Been
  - Declared.
    - And
      - If:

```
static class ListableClass{
    this.class for enum;
    public int intField;
    public int intReturner(...){...}
    public boolean boolReturner(){...}
    public void voidReturner(...){...}
    ⋮
}
```

- Then:

ListableClass

will

- Be
  - Listable,
    - ▹ Or:

ListableClass

will

- Be
  - A list
    - ▹ That:
      - Class,
- And

- Initially,
  - Its:
    - Length

will

- Be:

0.

- And so
  - Unless
    - And:
      - Until,

the

- Length
  - Of:

ListableClass

is

- Greater
  - Than:

0,

we

- Cannot
  - Access
    - Any:
      - Member

of

- That
    - Class.
        - And:
            - So

to

- Increase
    - Its:
        - Length,

we

- Execute:

$$\text{ListableClass.new}[\texttt{"}\text{someKey}\texttt{"}]; \tag{58}$$

- Or if
    - We
        - Execute:
            - Statement 58,

an

- Instance
    - Named:

$$\texttt{"}\text{someKey}\texttt{"}$$

will

- Created,
    - And
        - Then
            - Appended to:

ListableClass.

- And
  - So
    - ▹ If:

```
ListableClass.new["keyOne"];

ListableClass.new["keyTwo"];
```

two

- Instances
  - Will
    - ▹ Be:
      - – Created,
- And
  - There
    - ▹ Will:
      - – Be

a

- Separate
  - Database
    - ▹ For:
      - – Those

two

- Instances.
  - And

        ▷ After:
            – That,

we

- Can
    - Execute:

$$i = \text{ListableClass["keyOne"].intField} + \text{ListableClass["keyTwo"].intReturner(\ldots);}$$

- And
    - When
        ▷ We:
            – Start

the

- Program,
    - There
        ▷ Will:
            – Be

no

- Instance
    - In
        ▷ The:
            – Memory.
- Then
    - If:

string s = "someKey";

ListableClass.new[s];

- And
  - An
    - Instance
      - Using:

"someKey"

was

- Created
  - When
    - We:
      - Executed

the

- Program
  - Last:
    - Time,

then

- An
  - Instance
    - Using:

"someKey"

will

- Be

        ◦ Created,
            ▹ And:
                – The database

of

- That
    - Instance
        - ▹ Will:
            - Be loaded,

else

- A new
    - Instance,
        - ▹ And:
            - Also

a

- New
    - Database
        - ▹ For:
            - That instance

will

- Be
    - Created.
        - ▹ And
            - If:

```
ListableClass.new["sameKey"];

ListableClass.new["sameKey"];
```

the

- Program
    - Will
        - ▷ Ignore
            - The second:

$$\text{ListableClass.new["sameKey"];}$$

- And:

$$\text{string} \quad s \quad = \quad \text{class.key;} \tag{59}$$

can

- Be
    - Used
        - ▷ To:
            - Get

the

- Key
    - Of
        - ▷ The:
            - Instance.
- And
    - If:

$$\text{ListableClass["someKey"]} \quad == \quad \text{true,}$$

if

- An
    - Instance

▹ Named:

"someKey"

has

- Been
  - Loaded
    - In:
      - The memory,
- And:

(native)ListableClass["someKey"] == true,

if

- An
  - Instance
    - Named:

"someKey"

is

- Present
  - In
    - The:
      - Database.
- Assume
  - That
    - An instance
      - Named:

"instanceNotInMemory"

has

- Not
  - Been
    - ▷ Loaded:
      - In

the

- Memory.
  - Then:

```
ListableClass["instanceNotInMemory"].voidReturner(...);
```

will

- Throw
  - An:
    - ▷ Exception.
- But
  - We
    - ▷ Can
      - Write.

```
ListableClass.new["instanceNotInMemory"].voidReturner(...);
```

- Let:

sl

be

- An
  - Instance
    - ▷ Of:

$$[\text{string}].$$

- Then:

$$\text{sl} = \text{ListableClass}; \quad \textit{or} \quad \text{sl} = \text{ListableClass}[\,];$$

can

- Be
    - Used
        - To:
            - Get

the

- List
    - Of
        - All:
            - Keys

of

- All
    - Instances
        - In:
            - The memory,
- And:

$$\text{sl} = (\text{native})\text{ListableClass};$$

to

- Get
    - The
        - List:

      - Of

all

- Keys
    - Of
        - ▷ All:
            - Instances

in

- The
    - Database.
        - ▷ And:
            - We

can

- Write:

```
sl     =  ListableClass[ ] : (ListableClass[ ].boolReturner());

sl     =  (native)ListableClass[ ]
                    : ((native)ListableClass[ ].boolReturner());

int i  =  (ListableClass).length  +  ((native)ListableClass).length;
```

- And
    - If:

```
ListableClass.null["someKey"];        (60)
```

the

- Instance
    - Named:

$$\texttt{"someKey"}$$

will

- Be
  - Removed
    - ▷ From:
      - – The memory,
- And
  - All
    - ▷ Uncommitted:
      - – Things

in

- It
  - Will
    - ▷ Be:
      - – Lost.
- And
  - If:

$$\text{ListableClass.final}[\texttt{"someKey"}]; \tag{61}$$

then

- That
  - Instance
    - ▷ Will:
      - – Not only

be

- Removed

  - From
    - ▷ The:
      - Memory,
- But
  - Its
    - ▷ Database:
      - Will

also

- Be
  - Deleted.
    - ▷ And:
      - We

can

- Write:

ListableClass.null[this]; *and* ListableClass.final[this]; (62)

- And
  - Statements 58, 60, 61 and 62
    - ▷ Written
      - Inside:

ListableClass

will

- Be
  - Converted
    - ▷ To:

new["someKey"]; null["someKey"]; final["someKey"]; (63)

null[this]; *and* final[this]; (64)

respectively.

- Let:

```
static class ListableClass{
    this.class                             for      enum;
    void methodForNull(string)             for      null;
    void methodForFinal(string)            for      final;
    private void methodForNull(string s){...}
    protected void methodForFinal(string s){...}
    ⋮
}
```

- Then:

methodForNull("someKey");

will

- Be
  - Executed
    - Before:

ListableClass.null["someKey"];

- And:

methodForFinal("someKey");

- Before:

ListableClass.final["someKey"];

- Or

- Code
  - ▷ Generated
    - For:

ListableClass.null["someKey"];

will

- Be
  - Equivalent
    - ▷ To:

ListableClass["someKey"].methodForNull("someKey");

ListableClass.null["someKey"];

- And
  - Similarly,
    - ▷ For:

ListableClass.final["someKey"];

- And:

this.class for enum;

will

- Be
  - Inherited.
    - ▷ And
      - If:

SuperStaticClass

is

- Non
    - Listable,
        - And:

ListableClass *extends*: SuperStaticClass,

then

- The
    - Database
        - Of:

SuperStaticClass

will

- Not
    - Be
        - Listable
            - In:

ListableClass.

- And so
    - The
        - Database
            - Of:

SuperStaticClass

will

- Act
    - As:

            ▷ A meta
                – Database

of

- All
    - Users.
        ▷ And
            – If:

```
static   class   ListableClass{

    this.class     for     enum;

    static     native     int     someField;

    ⋮

    finally{. . .}

    ⋮

}
```

then

- There
    - Will
        ▷ Be
            – Only one:

someField

for

- Each
  - Element
    - In:

ListableClass.

- But:

```
static class NonListableStaticClass{
    static int someField;
}
```

is

- Equivalent
  - To:

```
static class NonListableStaticClass{
    int someField;
}
```

- And
  - If
    - We
      - Write:

class.length < 10; *or* this.class.length < 10;

in

- The
    - Property,
        - Then:
            - There

cannot

- Be
    - More
        - Than:
            - Ten instances

of

- That
    - Class.
        - And
            - The list:

ListableClass

will

- Be
    - Invisible
        - Outside:
            - The server.
- And
    - If:

```
static  class  StaticClass{
    public    native    [int]        k5;
    public    native    [int]        k6;
    public              [int  int]   view       =     (){...};
    void  methodOne((null)string,  int  int)    for    view.insert;
    private  void  methodOne((null)string  s,  int  int  t){...}
    ⋮
}
```

- Then:

```
StaticClass.view    +=    8  8,    9  9;
```

will

- Be
  - Converted
    - ▷ To:

```
[int  int]    k    =    8  8,    9  9;
for(int    i    in    0  ..  k.length)
    StaticClass.methodOne(class.key,  k[i]);
```

in

- Listables,

  ◦ And
    ▷ To:

```
[int int]   k   =   8 8,   9 9;

for(int   i   in   0 .. k.length)

    StaticClass.methodOne(null,  k[i]);
```

in

- Non
  ◦ Listables.
    ▷ And
      – If:

```
static class StaticClass{

    public   native   [int]   k5;

    void methodOne(int)    for   k5.insert;

    private void methodOne(int i1){...}

    ⋮

}
```

- Then:

```
StaticClass.k5   =     8;

StaticClass.k5   +=    9;
```

will

- Be
  - Converted
    - ▹ To:

```
StaticClass.k5 = ;

StaticClass.methodOne(8);

StaticClass.methodOne(9);
```

in

- Non
  - Listables
    - ▹ And:
      - Listables.
- But:

```
static class StaticClass{
    public native [int] k5;
    void methodOne(string, int) for k5.insert;
    private void methodOne(string s, int i1){...}
    ⋮
}
```

will

- Not
  - Compile.
    - Let:

NonListableClass

be

- Some:
  - Non
    - Listable
      - Class.
- Then:

```
[string]  sl  =  NonListableClass,  (native)NonListableClass;
```

will

- Be
  - Converted
    - To:

```
[string]  sl  =  <default-value>,  <default-value>;
```

- And
  - Statements 63 and 64
    - Written:
      - In

non

- Listables,
  - And:

this.class for enum;

written

- In:
    - Non
        - Static
            - Classes

will

- Be
    - Ignored,
        - And:

s == <default-value>,

if

- Statements 59
    - Are
        - Executed:
            - In

non

- Listables.
    - And:

```
[int int] k20;

(int i)k20 += 80 80;
```

will

- Be
    - Converted
        - To:

```
[int  int]    k20;

int           i;

k20    +=    80  80;
```

- But:

```
(int  i)[int  int]    k20    =    80  80;;
```

will

- Not
    - Compile.
        - And
            - If:

i

is

- Non
    - Native,
        - Then:

i.commit; *and* i.rollback;

will

- Be

    - Ignored.
        - ▷ Let:

```
int methodWithCommit(int i1, int i2){
    int i = 8, j;
    i += i1;
    j += i2;
    return.commit i + j;
}
```

- And
    - We
        - ▷ Execute:

```
int i3 = methodWithCommit(10, 20);
i3     = methodWithCommit(30, 40);
```

- And
    - When:

```
return.commit i + j;
```

- In:

```
int i3 = methodWithCommit(10, 20);
```

is

- Executed,
    - The
        - ▷ Values:
            - In

all

- Local
    - Variables
        - ▷ Will:
            - Be saved

in

- The
    - Memory,
        - ▷ And
            - When:

$$\text{int} \quad i \quad = \quad 8, \quad j;$$

- In:

$$i3 \quad = \quad \text{methodWithCommit}(30, \ \ 40);$$

is

- Executed,
    - Those
        - ▷ Values:
            - Which

where

- Saved
    - During

        - ▹ The:
            - Previous-execution

will

- Be
    - ◦ Restored
        - ▹ Instead
            - Of:

$$8 \quad \textit{and} \quad \text{<default-value>}.$$

- And
    - ◦ Similarly,
        - ▹ In:

```
int someMethod(int n){
    int i = n;
    if (i < 80) return.commit i++; else return i;
}
```

- And
    - ◦ Even
        - ▹ Though,
            - Statements like:

```
int i = j;
```

will

- Not

- Nullify
  - ▹ The effect,
    - Statements like:

$$i = 0; \quad \textit{and} \quad i = j;$$

will

- Nullify
  - It.
    - ▹ And:
      - Instances

of

- Classes
  - That
    - ▹ Implements:

Serializer

can

- Serialize
  - Any:
    - ▹ Object.
- And
  - Instances
    - ▹ Of:
      - Classes

that

- Implements:

Deserializer

can

- Deserialize
  - Any:
    - ▹ Object.
- And
  - So
    - ▹ If:

```
class   Writer   implements   Serializer{

    ⋮

}

class   Reader   implements   Deserializer{

    ⋮

}
```

we

- Can
  - Write:

```
SomeClass    sc1    =    . . . ,    sc2;

Writer       wr     =    . . . ;
```

```
sc1    >    wr;

Reader      re     =    ...;

sc2    <    re;
```

- And:

$$\text{statement}_1;$$

- In:

```
int   i,   j;

⋮

switch(i,  j){

    case   80 ,     :   statement_1;
                        break;

    case     ,  80  :   statement_2;
                        break;

    ⋮

}
```

will

- Be
  - Executed,
    - ▹ If:

$$\text{i} \;==\; 80,$$

ignoring

- The
    - Value
        - ▹ In:

j.

- And
    - Similarly,
        - ▹ In.

```
boolean   b1   =   bool-Exp_1;

boolean   b2   =   bool-Exp_2;

switch(b1,  b2){

    case    true,    true   :   Statement_1;
                                break;

    case    true,    false  :   Statement_2;
                                break;

    case    false,   true   :   Statement_3;
                                break;

    case    ,               :   ...;

}
```

2 Diagrams

Let:

$$G = (V, E),$$

- Where:

$$V = \{ q_1, q_2, q_3, q_4, q_5, q_6, \dots \},$$

$$E = \{ \langle q_1, q_3 \rangle, \langle q_1, q_4 \rangle, \langle q_2 \; q_5 \rangle, \langle q_2, q_6 \rangle, \dots \}.$$

- Then we see that,
  - Compilers
    - Can:
      - Generate

the

- State
  - Diagram
    - Equivalent
      - Of:

$$G$$

- From.

$$
\begin{array}{lll}
q_1 & = & \{ \\
 & & \quad (\text{bool-Exp}_1; \quad \text{some-Action}; \quad q_3), \\
 & & \quad (\text{bool-Exp}_2; \quad \ldots; \quad q_4), \\
 & & \quad \vdots \\
 & & \}; \\
 & & \\
q_2 & = & \{ \\
 & & \quad (\text{bool-Exp}_3; \quad \ldots; \quad q_5), \\
 & & \quad (\text{bool-Exp}_4; \quad \ldots; \quad q_6), \\
 & & \quad \vdots \\
 & & \}; \\
 & & \vdots
\end{array}
$$

- And
    - ○ So
        - ▷ We:
            - – First

present

- Edge
    - ○ Expressions,
        - ▷ And:
            - – Then groups,

which

- Encapsulates
    - The
        - ▹ Outgoing:
            - Edges.
- We
    - Will
        - ▹ Soon:
            - Show

how

- We
    - Represent
        - ▹ States.
            - Let:

state1

be

- Some
    - State.
        - ▹ Then:
            - An example

of

- An
    - Edge
        - ▹ Expression
            - Is:

(bool-Exp; simple-Statement; state1),

- And its
  - Interpretation
    - ▷ Is,
      - If:

bool-Exp

is

- Satisfied,
  - Then:

simple-Statement;

will

- Be
  - Executed,
    - ▷ And:
      - The object

will

- Go
  - To:

state1.

- The
  - Statement:

simple-Statement;

can

- Be
    - Any
        - ▹ Simple-statement
            - Except.

continue; *and* break;

- And
    - So:

```
i++;

int i = methodOne(...), j = methodTwo(...);

voidReturner();
```

can

- Be
    - Written
        - ▹ In:
            - Edge-expressions,
- But
    - Not.

for(...){...}

- The
    - Interpretation
        - ▹ Of:

(bool-Exp; ; state1),

is

- Similar,
    - Except
        - ▹ That:
            - Nothing

will

- Be
    - Executed
        - ▹ Before
            - Entering:

$$state1.$$

- And
    - If:

$$(bool\text{-}Exp;\quad someAction();),$$

- Then:

$$someAction();$$

will

- Be
    - Executed,
        - ▹ And:
            - There

will

- Be
    - No:

    - ▹ State
        - Transition.

- And:

$$(;;)$$

is

- Equivalent
    - To:

$$(\text{false};;).$$

- Let.

```
class SomeClass{
    public state state1, state2, state3;
    public SomeClass(){...}
    ⋮
}
```

- Then
    - To:
        - ▹ Define

the

- Out
    - Going
        - ▹ Edges

- Of:

state1, state2 *and* state3,

we

- Give
  - A group
    - For:
      - Each

of

- Them
  - In
    - Some:
      - Method.
- Exemplifying,
  - If
    - We
      - Write:

$$\text{state1} = \{ \text{Edge}_1, \text{Edge}_2 \};$$

in

- Some
  - Method,

▹ Then:

$$\text{Edge}_1 \qquad \textit{and} \qquad \text{Edge}_2$$

will

- Become
  - The
    - ▹ Out going
      - Edges of:

state1.

- And
  - So
    - ▹ To:
      - Implement

the

- State
  - Diagram,
    - ▹ We:
      - Give

a

- Group
  - To
    - ▹ All:
      - States

in

- Some method.

- But
  - ▹ Since:
    - – It

can

- Cause
  - The:
    - ▹ State
      - – Diagram

to

- Vary
  - At
    - ▹ Runtime:
      - – We

inform

- The
  - Compiler
    - ▹ That:
      - – Only groups

written

- In
  - A particular
    - ▹ Method:
      - – Should

be

- Used

        - To
            - Construct:
                - The diagram,

- And
    - Nothing
        - Else:
            - Should

be

- Used
    - For:
        - It.

- And
    - So
        - If:
            - Only

the

- Groups
    - Written
        - In:

$$\text{int} \quad \text{someMethod(float);} \tag{65}$$

should

- Be
    - Used
        - To:
            - Construct

the

- Public:
    - State
        - ▹ Diagram,

we

- Write:

int someMethod(float) for public;

in

- The
    - Class:
        - ▹ Body.
- Or
    - In
        - ▹ Doing:
            - – So,

first

- The
    - Compiler
        - ▹ Will:
            - – Ignore

all

- Groups,
    - And
        - ▹ Compile:
            - – The class.

- And
    - After
        - That:
            - Groups

written

- In
    - Method 65
        - Will:
            - Be used

to

- Construct
    - The:
        - Public
            - State-diagram.
- And so
    - If
        - The:
            - Groups

for

- All
    - Edges
        - Have:
            - Been

given

- In

- ◦ Both:

int methodOne(float); *and* int methodTwo(float);

- And
    - ◦ We
        - ▹ Wrote:

int methodOne(float) for public;

the

- Compiler
    - ◦ Will
        - ▹ First:
            - – Ignore

all

- Groups
    - ◦ In
        - ▹ Those:
            - – Two methods,
- And
    - ◦ After
        - ▹ That,
            - – Groups in:

int someMethod(float);

will

- Be
    - ◦ Used

▹ To:
  - Construct

the

- State
  - ○ Diagram.
    - ▹ And:
      - So

we see that,

- Groups
  - ○ Can
    - ▹ Be:
      - Written

in

- Any:

constructor *and* *method*,

- And
  - ○ It
    - ▹ Will:
      - Not matter

if

- They
  - ○ Are
    - ▹ Written
      - Inside:

an if-statement *or* *a for-loop* *or* *a while-loop*,

- And
  - They
    - ▹ Will:
      - Not

affect

- The
  - Execution
    - ▹ Of:
      - Methods.
- And so
  - If
    - ▹ We
      - Wrote:

$$\text{int someMethod(float)} \quad \text{for} \quad \text{public;} \tag{66}$$

there

- Will
  - Be
    - ▹ No:
      - Problem

if

- The
  - Action
    - ▹ Statement:
      - Of

an

- Edge
  - Invokes.

$$\text{int}\quad \text{someMethod(float);}$$

- But
  - Local
    - ▹ Variables:
      - Cannot

be

- Used
  - In:
    - ▹ Edge
      - Expressions.
- And
  - We
    - ▹ Can
      - Write.

```
state1   +=   {...};
state1   =    this.state1   +   {...};
state1   =    state1        +   {...};
state1   +=   state2        +   {...};
```

- And
  - So

▷ If:

state1 $\mathrel{+}=$ $\{$ Edge$_1$; $\}$;

state1 $=$ $\{\}$;

initially,

- One
  - ○ More
    - ▷ Edge:
      - – Will

be

- Added
  - ○ To:

state1,

- And
  - ○ Then
    - ▷ All:
      - – Its edges

will

- Be
  - ○ Removed.
    - ▷ And
      - – So:

state1 $=$ Edge$_1$;

state2 $\mathrel{+}=$ Edge$_2$;

should

- Be
  - Rewritten
    - ▷ As:

$$\begin{array}{llll} \text{state1} & = & \{ & \text{Edge}_1 & \}; \\ \text{state2} & += & \{ & \text{Edge}_2 & \}; \end{array}$$

- And
  - It
    - ▷ Is:
      - Not mandatory,

that

- Groups
  - Be
    - ▷ Given:
      - To

all

- States.
  - And
    - ▷ Since:
      - The signature

of

- Methods
  - And

            - ▹ Fields:
                - Is fixed,

- And
    - ○ Since
        - ▹ The:
            - Syntax

of

- Boolean
    - ○ Expressions
        - ▹ Is:
            - Fixed,

we see that,

- We
    - ○ Can
        - ▹ Check:
            - Whether

the

- Satisfaction
    - ○ Of:
        - ▹ A boolean
            - Expression

implies

- The
    - ○ Satisfaction
        - ▹ Of:
            - Another,

without

- Checking
    - For:
        - Method
            - Equivalence.
- And
    - So
        - There:
            - Will

be

- An
    - Error
        - In:

$$\text{state1} = \{ \\ (\text{bool-Exp}_1;\quad \ldots;\quad \ldots), \\ (\text{bool-Exp}_2;\quad \ldots;\quad \ldots) \\ \};$$

if

- The
    - Satisfaction
        - Of:

$$\text{bool-Exp}_1$$

implies

- The
  - Satisfaction
    - ▹ Of:

$$\text{bool-Exp}_2.$$

- Let:

sc

be

- An
  - Instance
    - ▹ Of:

SomeClass.

- Then
  - If:

```
sc.state = state1;
sc.state++;
```

- And
  - When:

```
sc.state = state1;
```

is

- Executed,

    - The
        - ▹ State
            - Of:

sc

will

- Become:

state1,

- And

    - When:

sc.state++;

is

- Executed:

sc

will

- Search

    - Through
        - ▹ All:
            - Edges

in

- The

    - Group
        - ▹ For:

state1,

- And
    - Make
        - A transition:
            - Using

the

- First
    - Edge
        - Whose:
            - Boolean-expression

is

- Satisfied.
    - Note that,
        - We:
            - Did

not

- Write:

$$\text{sc.state} = \text{SomeClass.state1};$$

since

- The
    - Compiler
        - Can
            - Understand that:

$$\text{state1}$$

is

- Allowed

- For:

  sc.state.

- And
  - We
    - Can
      - Write:

```
if (sc.state == state1){...}
if (sc.state in (state1, state2)){...}
if (sc.state !in (...)){...}
string string1 = sc.state;
```

- Outside:

  SomeClass,

- And:

```
public.state = state1;
public.state++;
if (public.state != ...){...}
if (public.state in (...)){...}
if (public.state !in (...)){...}
string string1 = public.state;
```

- Inside:

SomeClass.

- And
    - There
        - ▹ Will:
            - Be

no

- Change
    - In:

<object-name>.state,

- If:

$$< \text{object-name} > .\text{state} + +; \tag{67}$$

throws

- An exception.
    - And
        - ▹ Similarly,
            - For:

$$\text{public.state} + +; \tag{68}$$

- And
    - There
        - ▹ Will:
            - Be

an

- Exception,

    ◦ If
        ▷ The:
            – Action-statement

of

- An
    ◦ Edge
        ▷ Tries:
            – To make

a

- Transition.
    ◦ Let:

$$\mathrm{sc1} \quad \textit{and} \quad \mathrm{sc2}$$

be

- Instances
    ◦ Of:

$$\mathrm{SomeClass.}$$

- Then
    ◦ We
        ▷ Do not
            – Allow:

$$\mathrm{sc1.state} = \mathrm{sc2.state;}$$

- But
    ◦ We
        ▷ Can

- Write:

$$\text{sc1.state} \quad == \quad \text{sc2.state} \qquad \textit{and} \qquad \text{sc1.state} \quad != \quad \text{sc2.state}. \quad (69)$$

- And
  - If:

```
(public)state               tps1    =     public.state;
[(public)state]             k27     for   tps1;
public.state                        =     tps1;
(protected.static)state     tpros   =     ...;
[int (public)state]         k28;
k28                                 +=    8 public.state;
```

- Then:

tps1

can

- Store
  - The
    - ▹ Value
      - Of:

public.state,

- Or
  - Public

        ▹ States:
            – Defined

in

- The
    - Class.
        ▹ And so
            – If:

$$\text{ct1} \quad \textit{and} \quad \text{ct2}$$

are

- Instances
    - Of:

$$\begin{aligned}
&\text{class ClassTwo}\{\\
&\quad \text{private} \quad \text{state} \qquad\qquad\qquad \ldots;\\
&\quad \text{public} \quad (\text{private})\text{state} \quad \text{tPrvs} \quad = \quad \ldots;\\
&\quad \vdots\\
&\}
\end{aligned}$$

then

- We
    - Cannot
        ▹ Write:

$$\text{ct1.tPrvs} \quad = \quad \text{ct2.tPrvs};$$

- But
    - We
        - ▹ Can
            - Write:

$$\text{if} \quad (\text{ct1.tPrvs} \quad == \quad \text{ct2.tPrvs})\{\ldots\}$$

- And
    - If:

$$\text{ts1} \qquad\qquad \textit{and} \qquad\qquad \text{ts2}$$

are

- Of
    - Type:

$$\text{(public)state,}$$

then

- We
    - Cannot
        - ▹ Write:

$$\text{ts1} \quad = \quad \text{ts2;}$$

- But
    - We
        - ▹ Can
            - Write:

$$\text{if} \quad (\text{ts1} \quad == \quad \text{ts2})\{\ldots\}$$

- And

- Statements
  - ▹ Like:
    - – Statements 66 and 68,

- And:

(public)state ts;

will

- Not
  - Compile
    - ▹ In:
      - – Classes

with

- No:
  - Public
    - ▹ States,
- And
  - Similarly:
    - ▹ For statement 67
      - – And expressions 69.
- And
  - Non-static
    - ▹ Public-edges
      - – In:

```
class  ClassOne{
    public                  state   ...;
    protected               state   ...;
    private                 state   ...;
    public       static     state   ...;
    protected    static     state   ...;
    private      static     state   ...;
    ⋮
}
```

cannot

- Point
  - To:

a private-state *or* *a protected-state* *or* *a static-state,*

- But
  - Only:
    - ▹ To

non

- Static
  - Public:
    - ▹ States.
- And

  - ◦ Similarly,
    - ▹ For:
      - – Others.
- And
  - ◦ We
    - ▹ Can
      - – Write:

$$\text{protected.state} = \ldots;$$

$$\text{private.state} = \ldots;$$

$$\text{private.static.state} = \ldots;$$

$$\text{protected.state++};$$

$$\text{private.state++};$$

$$\text{private.static.state++};$$

inside

- That
  - ◦ Class,
    - ▹ And:

$$\text{ClassOne.state} = \ldots;$$

$$\text{ClassOne.state++};$$

- And

- Expressions
  - Like:

    ClassOne.state == . . .

    ClassOne.state in (. . .)

outside

- That
  - Class.
    - And:
      - It

is

- Possible
  - To
    - Give:
      - Groups

for

- Non-static
  - Public
    - State-diagram
      - In:

a private-method or a protected-method or a static-method.

- But
  - Since
    - It:
      - Can confuse,

we

- Say
    - That,
        - ▹ If:

$$\text{void}\quad \text{someMethod()}\qquad \text{for}\qquad \text{public;} \tag{70}$$

- Then:

$$\text{void}\quad \text{someMethod();}$$

should

- Be:
    - Public
        - ▹ Non
            - Static.
- And
    - We
        - ▹ Say:
            - That

there

- Will
    - Be
        - ▹ An:
            - Error,

if

- The
    - Group:
        - ▹ Of

a

- Private
    - State
        - ▹ Is:
            - Written

in

- A public
    - Method.
        - ▹ And:
            - Similarly,

for

- Others.
    - And
        - ▹ If:

```
public static void staticMethod();
```

is

- Some
    - Static
        - ▹ Method:
            - We

can

- Write:

```
void staticMethod() for public;
```

for

- The

- Public
  - ▷ Static:
    - State-diagram.
- But:

```
static  void  staticMethod()  for          public;

static  void  staticMethod()  for   public.static;

void          staticMethod()  for   public.static;
```

will

- Not
  - Produce
    - ▷ Any:
      - Error.
- And
  - Statements
    - ▷ Like:
      - Statement 70

cannot

- Be:

public *or* protected *or* private.

- And:

```
final  void  someMethod()  for  public;
```

is

- Equivalent

    - To:

$$\text{void someMethod() for public;}$$

- And
    - We
        - Can:
            - Say

that,

- If:

$$\text{void methodOne(), void methodTwo() for public;} \tag{71}$$

the

- Compiler
    - Will
        - First:
            - Execute

all

- Groups
    - In:

$$\text{void methodOne();}$$

- And
    - Then
        - That
            - In:

$$\text{void methodTwo();}$$

so that,

- We
    - Can
        - ▷ Avoid:
            - Congestion.
- But we see that,
    - If
        - ▷ We
            - Write:

```
void  methodOne(){
     ⋮
}
void  methodTwo(){
     ⋮
}
```

we

- Will
    - Still
        - ▷ Avoid:
            - Congestion.
- And
    - Then
        - ▷ If:
            - We remove

the

- Last:

  ‘}’

- Of:

  void methodOne();

- And:

  “void methodTwo(){”

- Of:

  void methodTwo();

we

- Will
  - Still
    - ▷ Avoid:
      - Congestion.
- And
  - So
    - ▷ We see that:
      - There

is

- No
  - Gain
    - ▷ In:
      - Splitting it.
- And

- So
    - We:
        - Do

not

- Allow
    - Statements
        - Like:
            - Statement 71,
- And
    - Only
        - One:
            - Statement

can

- Be
    - Used
        - For:
            - For-ing.
- Let:

subSys1

be

- A subsystem
    - Of:

state1,

- And
    - Let:

subState1 *and* subState2

be

- Two
  - States
    - Of:

subSys1.

- Then
  - We
    - Write.

state1["subSys1"] subState1, subState2;

- Note that,
  - We
    - Did not
      - Declare:

subSys1,

since

- When
  - We
    - Write:

state1["subSys1"] . . . ;

the

- Compiler
  - Can

        ▷ Recognize
            – That:

$$\text{subSys1}$$

is

- A subsystem
    - Of:

$$\text{state1}.$$

- And
    - Since:

$$\text{state1}$$

- Is:

$$\text{public},$$

- We see that:

$$\text{public}$$

is

- Included
    - In:

$$\text{state1["subSys1"]}.$$

- And so
    - We
        ▷ Cannot
            – Write:

$$\text{protected} \quad \text{state1["subSys1"]} \quad \ldots;$$

- But:

```
public    state1["subSys1"]    ...;
```

will

- Not
  - Produce
    - Any:
      - Error.
- And
  - Subsystem
    - Names:
      - Should

be

- Unique.
  - And
    - So:

```
state1["sameName"]        subState1,    ...;

subState1["sameName"]    ...;
```

will

- Not
  - Compile.
    - But
      - We allow:

```
sameStateName["sameStateName"]    ...;
```

- And to
    - Make
        - ▹ Transitions
            - – In:

$$\text{subSys1},$$

we

- First
    - Enter
        - ▹ Into:

$$\text{state1},$$

- And
    - Execute
        - ▹ Edges
            - – Like:

$$(\text{bool-Exp};\quad \text{someAction}();\quad \text{state1}[\texttt{"subSys1"}]++), \tag{72}$$

which

- Would
    - Have
        - ▹ Been:
            - – Written

in

- The
    - Group
        - ▹ For:

state1.

- When
  - Edge 72
    - ▹ Is executed,
      - First:

someAction();

will

- Be
  - Executed,
    - ▹ And:
      - The system

will

- Remain
  - In:

state1,

- And
  - Transition
    - ▹ Will:
      - Be

made

- In:

subSys1.

- Then
  - If

            ▹ We
                - Execute:

$$(\ldots; \quad \ldots; \quad \text{state.outer++}),$$

the

- State
    - Of:

$$\text{subSys1}$$

will

- Be
    - Saved,
        - ▹ And:
            - Transition

will

- Be
    - Made:
        - ▹ In

the

- Outer
    - Layer.
        - ▹ And:
            - If

we

- Enter:

$$\text{state1},$$

- And
    - Execute:

$$(\ldots; \quad \ldots; \quad \text{state1}[\texttt{"}\text{subSys1}\texttt{"}]),$$

then

- No
    - Transition
        - Will:
            - Be

made

- In:

$$\text{subSys1},$$

- But
    - Only
        - Control:
            - Will

be

- Passed
    - To:
        - It.
- And
    - So
        - No:
            - Transition

will

- Be
  - Made
    - In:

$$\text{subSys1},$$

until

- We
  - Execute:

$$\text{public.state++;}$$

- And
  - Similarly,
    - If
      - We execute:

$$(\ldots; \quad \ldots; \quad \text{state.outer}), \tag{73}$$

no

- Transition
  - Will
    - Be:
      - Made

in

- The
  - Outer:
    - Layer,
- But
  - Only

- ▷ Control:
  - Will

be

- Passed
  - To:
    - ▷ It.
- And
  - So:
    - ▷ Substate
      - Edges

cannot

- Point
  - To
    - ▷ States:
      - Outside

its

- Circle.
  - And
    - ▷ Statements
      - Like:

$$\text{subState1} \quad = \quad \{\ldots\};$$

can

- Be
  - Used
    - ▷ To:

- Give edges,

- And:

$$\text{state1}[\texttt{"subSys1"}] \quad = \quad \text{subState1};$$

for

- Initialization.
  - Let:

$$\text{subSys2} \qquad \textit{and} \qquad \text{subSys2}$$

be

- Be
  - Two
    - ▷ Subsystems
      - Of:

$$\text{state1}.$$

- Then
  - If
    - ▷ We
      - Execute:

$$(\ldots; \quad \ldots; \quad \text{state1}[\texttt{"subSys1"}]{+}{+} \quad \&\& \quad \text{state1}[\texttt{"subSys2"}]{+}{+}),$$

first

- Trasnsition
  - Will
    - ▷ Be
      - Made in:

$$\text{subSys1},$$

- And
    - Then
        - ▷ In:

subSys2.

- And when
    - The next
        - ▷ Event:
            - Occurs,

first

- Trasnsition
    - Will
        - ▷ Be
            - Made in:

subSys1,

- And
    - Then
        - ▷ In:

subSys2.

- And
    - After
        - ▷ That,
            - If:

$$(\ldots; \quad \ldots; \quad \text{state.outer} + +) \tag{74}$$

was

- Executed
    - In:

subSys1,

then

- Transition
    - Will
        - ▷ Be:
            - Made

in

- The
    - Outer
        - ▷ Layer:
            - Only

after

- Making
    - A transition
        - ▷ In:

subSys2,

since

- We
    - Entered
        - ▷ Those:
            - Subsystems

by

- Executing:

$$\text{state1}[\texttt{"subSys1"}]{+}{+} \quad \&\& \quad \text{state1}[\texttt{"subSys2"}]{+}{+}.$$

- But
  - If
    - ▷ We
      - Execute:

$$(\ldots; \quad \ldots; \quad \text{state.goto.outer}++) \tag{75}$$

- In:

subSys1,

then

- Transition
  - Will
    - ▷ Be:
      - Made

in

- The
  - Outer
    - ▷ Layer:
      - Without making

any

- Transition
  - In:

subSys2.

- And

- Similarly,
  - ▹ For:

$$(\ldots; \quad \ldots; \quad \text{state.goto.outer}). \tag{76}$$

- And
  - If
    - ▹ We
      - Execute:

$$(\ldots; \quad \ldots; \quad \text{state.break}) \tag{77}$$

- In:

$$\text{subSys1},$$

- Then:

$$\text{subSys1},$$

will

- Relinquish
  - Control.
    - ▹ But:
      - Transition

will

- Be
  - Made
    - ▹ In:

$$\text{subSys2}$$

until

- We

- Execute
  - ▹ Edges:
    - Like edges 73 or 74 or 75 or 76.

- And
  - If
    - ▹ No:
      - Other subsystem

has

- Control,
  - Then
    - ▹ Edge 77:
      - Will

be

- Equivalent
  - To:
    - ▹ Edge 73.
- And
  - So
    - ▹ We
      - Allow:

$$(\ldots; \quad \ldots; \quad \text{state.break++}).$$

- Let:

ct

be

- An

- Instance
    - Of:

$$\begin{array}{llll} \text{class} \ \ \text{ClassThree}\{ & & & \\ \quad \text{public} \ \ \text{state} & \text{state1}, & \text{state2}, & \ldots; \\ \quad \text{state1}[\texttt{"subSys1"}] & \text{subState1}, & \text{subState2}, & \ldots; \\ \quad \text{state1}[\texttt{"subSys2"}] & \ldots; & & \\ \quad \text{subState1}[\texttt{"subSubSys"}] & \ldots; & & \\ \quad \vdots & & & \\ \} & & & \end{array}$$

- And
    - We
        - Execute:

$$\text{boolean} \quad b \quad = \quad \text{ct.state1}[\texttt{"subSys1"}];$$

- Then:

$$b \ == \ \text{true},$$

- If:

$$\text{ct.state} \ == \ \text{state1},$$

- And
    - If:

subSys1

has

- Received
  - Control.
    - ▹ And
      - If:

$$[\text{string}] \quad \text{sl} \quad = \quad \text{ct.state1}; \tag{78}$$

- Then:

sl

will

- Hold
  - The
    - ▹ Names:
      - Of

all

- Subsystems
  - That
    - ▹ Received:
      - Control,
- And
  - The
    - ▹ Order
      - In:

sl

will

- Be

    - The
        - Order:
            - In

which

- Those
    - Subsystems
        - Received:
            - Control.
- And
    - If:

$$\text{(enum)string} \quad \text{hierarchy} \quad = \quad \text{ct.state1;} \tag{79}$$

- Then:

hierarchy

will

- Hold
    - The
        - Hierarchy:
            - Of subsystems

that

- Received
    - Control,
        - And:

$$\text{hierarchy}[0] \quad == \quad \text{state1.}$$

- And

- ◦ We
    - ▹ Can
        - – Write:

$$[\text{string}] \quad \text{sl} \quad = \quad \text{state1}; \tag{80}$$

- Inside:

SomeClass.

- And
    - ◦ If:

$$\text{public.state} \;\; != \;\; \text{state1} \qquad \textit{or} \qquad \text{sc.state} \;\; != \;\; \text{state1},$$

- And
    - ◦ We execute
        - ▹ Statement 80 or 78 or 79,
            - – Then:

$$\text{sl.length} \;\; == \;\; 0 \qquad \textit{and} \qquad \text{hierarchy.length} \;\; == \;\; 0.$$

- And
    - ◦ We
        - ▹ Can
            - – Write:

$$\begin{aligned} \text{boolean} \quad \text{b} \quad &= \quad \text{ct.state1}[\texttt{"subSys1"}] \;\; == \;\; \ldots; \\ \text{b} \quad &= \quad \text{ct.state1}[\texttt{"subSys1"}] \;\; !\text{in} \;\; (\ldots); \end{aligned}$$

- And
    - ◦ If:

```
class ClassFour{
    public      static   state   sPublState1,   ...;
    sPublState1["sPublSubSys"]   ...;
    protected   static   state   ...;
    private     static   state   sPrvState1,    ...;
    sPrvState1["sPrvSubSys"]     ...;
    ⋮
}
```

we

- Can
  - Write:

    ```
    ClassOne.sPublState1["sPublSubSys"]  =  ...;
    ```

- And
  - Expressions
    - ▹ Like:

```
ClassOne.sPublState1["sPublSubSys"]     !=      ...
ClassOne.sPublState1["sPublSubSys"]     !in     (...)
```

- Outside:

  ClassOne,

- And:

```
if   (publState1["subSys"].push? == 1)   publState1["subSys"].push;
```

- Inside.

```
class  ClassFive{

    public   state          publState1,    . . . ;

    publState1["subSys"]    . . . ;

    public   [(publState1["subSys"])state]   subStack
                            for   publState1["subSys"];

    ⋮

}
```

- Assume
  - That
    - We
      - Had written:

```
state1.base      =   {

                         statement_1;

                         statement_2;

                     };

state1.default   =   {. . .};

state1.enter     =   {. . .};
```

$$\text{state1.exit} \quad = \quad \{\ldots\};$$

in

- The
  - Method
    - ▹ That:
      - Has

been

- For-ed.
  - And
    - ▹ Let:

$$\text{public.state} \quad == \quad \text{state1,}$$

- And
  - We
    - ▹ Execute:

$$\text{public.state++;}$$

- Then:

$$\text{state1.base}$$

will

- Be
  - Executed:
    - ▹ Before

the

- Program

- Scans:
  - Through

the

- Edges
  - Of:

state1.

- And
  - If
    - None:
      - Of

the

- Edges
  - Fires,
    - Then:

state1.default

will

- Be
  - Executed.
    - And:
      - If

we

- Leave:

state1,

- Then:

state1.exit

will

- Be
    - Executed
        - ▹ Before:
            - – Entering

the

- New
    - State.
        - ▹ And:
            - – If

we

- Reenter:

state1,

- Then:

state1.enter

will

- Be
    - Executed.
        - ▹ Note that:
            - – If

we

- Enter:

state1,

- Then:

state1.enter

will

- Be
    - Executed,
        - And:

state1.base

will

- Not
    - Be:
        - Executed.
- And
    - After
        - That:
            - If

we

- Execute:

public.state++;

- Then:

state1.base

will

- Be
    - Executed,

            ▷ And:

$$\text{state1.enter}$$

will

- Not
    - Be:
        - ▷ Executed.
- And
    - We
        - ▷ Can
            - Write:

$$\begin{aligned}
\text{public.base} &= \{\ldots\}; \\
\text{public.default} &= \{\ldots\}; \\
\text{public.enter} &= \{\ldots\}; \\
\text{public.exit} &= \{\ldots\};
\end{aligned}$$

for

- The
    - Public:
        - ▷ State
            - Diagram,
- And:

$$\begin{aligned}
\text{state1}[\texttt{"subSys1"}].\text{base} &= \{\ldots\}; \\
\text{state1}[\texttt{"subSys1"}].\text{default} &= \{\ldots\}; \\
\text{state1}[\texttt{"subSys1"}].\text{enter} &= \{\ldots\}; \\
\text{state1}[\texttt{"subSys1"}].\text{exit} &= \{\ldots\};
\end{aligned}$$

for

- Subsystems.
  - And
    - ▷ If:
      - We

are

- Presently
  - In:

subState1,

first

- The
  - Base
    - ▷ Of:

“*the state-diagram*”

will

- Be
  - Executed,
    - ▷ Followed:

- By

that

- Of:

state1,

- And
    - Then
        - ▹ That
            - Of:

subState1.

- And
    - First,
        - ▹ The default
            - Of:

subState1

will

- Be
    - Executed,
        - ▹ Followed:
            - By

that

- Of:

state1,

- And
    - Then

        ▷ That
            – Of:

"*the state-diagram.*"

- And
    - We
        ▷ Can
            – Write:

$$\text{state.break;} \qquad \textit{or} \qquad \text{state.break}++; \tag{81}$$

$$\textit{or}$$

$$\text{state.goto.outer;} \qquad \textit{or} \qquad \text{state.goto.outer}++; \tag{82}$$

in

- The:
    - Base
        ▷ Or
            – Default

of

- Subsystems.
    - And
        ▷ Statements 81 and 82:
            – Written

in

- Other
    - Places
        ▷ Will:
            – Be ignored.

- And
    - These
        - Blocks:
            - Will

be

- Executed,
    - Only
        - If:
            - They

are

- Written
    - The
        - Method:
            - That

has

- Been
    - For-ed,
        - And:

$$+=$$

can

- Be
    - Used
        - With:
            - Them.
- Let:

$$\text{SubClass} \quad \textit{extends}: \quad \text{SuperClass}.$$

- Then
  - The
    - ▹ State-diagram
      - Of:

$$\text{SuperClass}$$

will

- Be
  - Inherited
    - ▹ By:

$$\text{SubClass}.$$

- Let:

$$\text{stateOfSuperClass} \quad \textit{and} \quad \text{stateOfSubClass}$$

be

- States
  - Of:

$$\text{SuperClass} \quad \textit{and} \quad \text{SubClass}$$

$$\textit{respectively}.$$

- And
  - If:

$$\text{stateOfSubClass} \quad = \quad \text{stateOfSuperClass} \quad + \quad \{\ldots\}; \qquad (83)$$

- And
  - Either:

stateOfSubClass *or* stateOfSuperClass

- Is:

public,

then

- Both
  - Of
    - ▹ Them
      - – Should be:

public.

- And
  - Statements
    - ▹ Like:
      - – Statement 83

should

- Be
  - Written:
    - ▹ In

the

- Method
  - Of:

SubClass

that

- Has

    - Been
        - For-ed:
            - For

the

- State
    - Diagram.
        - Or:
            - Even

though,

- Some
    - Method
        - Of:

SuperClass

has

- Been
    - For-ed,
        - Groups
            - In:

SubClass

will

- Not
    - Be
        - Executed:
            - Unless

we

- Write
  - The
    - ▹ For-ing statement
      - In:

SubClass.

- And
  - So
    - ▹ If:

void someMethod() for public;

was

- Written
  - In:

SuperClass,

- Then:

void someMethod() for public;

or

void someOtherMethod() for public;

can

- Be
  - Written
    - ▹ In:

SubClass.

- And

- If
  - We
    - Write:

$$\text{stateOfSuperClass} \quad = \quad \{ \quad \text{Edge}_1 \quad \};$$

- In:

$$\text{SuperClass,}$$

- And:

$$\text{stateOfSuperClass} \quad += \quad \{ \quad \text{Edge}_2 \quad \};$$

- In:

$$\text{SubClass,}$$

- And:

$$\text{Edge}_1 \qquad \textit{and} \qquad \text{Edge}_2$$

have

- The
  - Same:
    - Boolean
      - Expressions,

then

- We
  - Can
    - Say
      - That:

$$\text{Edge}_2$$

will

- Override:

$$\text{Edge}_1.$$

- But
    - Since
        - ▹ It:
            - Can

produce

- Unexpected
    - Behaviors,
        - ▹ We:
            - Say

that

- The
    - Compiler
        - ▹ Will:
            - Signal

an

- Error
    - If
        - ▹ It:
            - Is so.
- But
    - If
        - ▹ We

- Write:

$$\text{stateOfSuperClass} \quad += \quad \{ \quad \text{(switch)Edge}_2 \quad \};$$

- And
  - The boolean
    - ▷ Expressions
      - Of:

$$\text{Edge}_1 \qquad \textit{and} \qquad \text{Edge}_2$$

are

- The
  - Same,
    - ▷ Then:

$$\text{Edge}_2$$

will

- Override:

$$\text{Edge}_1.$$

- And
  - If:

$$\text{stateOfSuperClass} \quad = \quad \{ \quad \text{(final)Edge}_1 \quad \};$$

- Then:

$$\text{Edge}_1$$

cannot

- Be
  - Overridden.

    - ▹ And:
        - Edges

given

- In:

$$\text{someState} \quad = \quad (\text{final})\{\ldots\};$$

$$\text{someState} \quad += \quad (\text{final})\{\ldots\};$$

cannot

- Be
    - ◦ Overridden.
        - ▹ And
            - If:

public final state1, state2;

then

- New
    - ◦ Edges
        - ▹ Cannot:
            - Be

given

- To:

state1 *and* state2

in

- Subclasses.

- And:

$$\text{state1} \quad = \quad \{ \quad \text{(partial)Edge} \quad \};$$

is

- Equivalent
  - To:

$$\text{state1} \quad = \quad \{ \quad \text{Edge} \quad \};$$

- And
  - Similarly,
    - For:
      - Others.
- And
  - If
    - We
      - Write:

$$\text{public.base} \quad = \quad \{\ldots\}; \tag{84}$$

- In:

$$\text{SuperClass,}$$

- And:

$$\text{public.base} \quad += \quad \{\ldots\};$$

- In:

$$\text{SubClass,}$$

then

- Statements

- ◦ Will
  - ▹ Be:
    - – Added

into

- The
  - ◦ Base
    - ▹ Inherited
      - – From:

$$\text{SuperClass.}$$

- But
  - ◦ If
    - ▹ We
      - – Write:

$$\text{public.base} \quad = \quad \{\ldots\};$$

- In:

$$\text{SubClass,}$$

then

- Statements
  - ◦ Inherited
    - ▹ From:

$$\text{SuperClass}$$

will

- Be
  - ◦ Removed.

        ▷ And
            – If:

$$\text{public.base} \quad = \quad (\text{partial})\{\ldots\};$$

- Then:

public.base

should

- Be
    - Extended
        ▷ In:
            – Subclasses.
- And
    - To
        ▷ Avoid:
            – Errors,

we

- Say
    - That,
        ▷ The
            – First:

$$\text{public.base} \quad += \quad \{\ldots\};$$

written

- In:

SuperClass

will

- Be
  - Equivalent
    - ▹ To:
      - Statement 84,
- And:

$$\text{public} \quad \text{final} \quad \text{final} \quad \text{state} \quad \ldots;$$

- To:

$$\text{public} \quad \text{final} \quad \text{state} \quad \ldots;$$

- And
  - Similarly,
    - ▹ For:
      - Others.
- And
  - All:
    - ▹ States
      - And substates

should

- Be
  - Initialized:
    - ▹ In

the

- Constructor
  - Or:
    - ▹ Static
      - Block.

- And
    - All:
        - ▷ Temporary
            - State-variables

should

- Be
    - Initialized
        - ▷ When:
            - Declared,
- And
    - States
        - ▷ Cannot:
            - Be committed

into

- The
    - Database.
        - ▷ And
            - So:

```
static class SomeSchema{
    public native [(public)state] k28;
    ⋮
}
```

will

- Not
  - Compile.
    - And:
      - We

do

- Not
  - Allow
    - Methods
      - Like:

state someMethod(state);

- And:

trees *and* *arrays*

- Of:

state.

- And
  - States
    - Written:
      - In interfaces

will

- Be
  - Ignored,
    - And:
      - Will

not

- Be
  - Passed
    - To:
      - Classes.
- Let:

i *and* j

be

- Of
  - Type:

int,

- And
  - Let:

string1 *and* matchValue

be

- Instances
  - Of:

string.

- Then
  - In:

```
for(i,    j;    string1;     matchValue){

        case    reg-Exp$_1$   :    statement$_1$;
                                   continue;

        case    reg-Exp$_2$   :    statement$_2$;
                                   continue;

        ⋮

}
```

the

- Program
  - ○ Will
    - ▹ Scan
      - – From:

“*the* i^{th} *character*”

- Of:

string1

- To:

“*the* $\mathrm{j} - 1^{th}$ *character*,”

- And
  - ○ When
    - ▹ A substring:
      - – Satisfies

a

- Regular
    - Expression,
        - ▹ That:
            - Substring

will

- Be
    - Stored
        - ▹ In:

$$\text{matchValue},$$

- And
    - The
        - ▹ Corresponding:
            - Statements

will

- Be
    - Executed.
        - ▹ And
            - If:

$$\text{string1.length} < j,$$

- Then:

$$\text{string1.length}$$

will

- Be
    - Used

            - ▹ Instead
                - Of:

$$\text{j.}$$

- And
    - In:

$$\text{for(i;}\quad\text{string1;}\quad\text{matchValue)}\{\ldots\}$$

the

- Program
    - Will
        - ▹ Scan
            - From:

$$\text{“}\textit{the}\quad \text{i}^{th}\quad \textit{character}\text{”}$$

till

- The
    - End.
        - ▹ And
            - In:

$$\text{for(;string1;}\quad\text{matchValue)}\{\ldots\}$$

the

- Program
    - Will
        - ▹ Scan
            - The whole:

$$\text{string1.}$$

- And
    - In:

$$\text{for(;string1;)}\{\dots\}$$

the

- Matched
    - Substring
        - Cannot:
            - Be used.
- And
    - If:

$$\text{for(}\dots;\quad\dots;\quad\text{string}\quad\text{matchValue,}\quad\text{int}\quad\text{i)}\{\dots\}\tag{85}$$

- Then:

$$\text{i}$$

will

- Hold
    - The
        - Starting:
            - Position

of

- The
    - Substring
        - That:
            - Was satisfied.
- And:

$$\text{for}(\ldots;\quad \ldots;\quad \text{int}\quad \text{i},\quad \text{string}\quad \text{matchValue})\{\ldots\}$$

is

- Equivalent
  - To:
    - ▷ Statement 85.
- And
  - We
    - ▷ Do not
      - Write:

$$\text{for}(\ldots;\quad \ldots;\quad \text{int}\quad \text{i};\quad \text{string}\quad \text{matchValue})\{\ldots\}$$

since

- We
  - Do
    - ▷ Not
      - Write:

$$\text{void}\quad \text{someMethod}(\text{int}\quad \text{i};\quad \text{string}\quad \text{s})\{\ldots\}$$

- And
  - When
    - ▷ We
      - Exit:

$$\text{for}(;\text{string1};\quad \text{matchValue},\quad \text{i})\{\ldots\}$$

the

- Last
  - Part

        - ▹ Of:

string1

that

- Did
    - Not:
        - ▹ Satisfy

any

- Regular-expression
    - Will
        - ▹ Be
            - Stored in:

matchValue,

- And:

i

will

- Store
    - The
        - ▹ Starting:
            - Position

of

- That:
    - Last
        - ▹ Part.
- And

- If
    - There:
        - Is

a

- Default
    - Area,
        - The:
            - Program

will

- Enter
    - There,
        - If:
            - It

enters

- The
    - Sink
        - State,
            - And:

matchValue

will

- Hold
    - The
        - Substring:
            - That

did

- Not
    - Satisfy
        - ▹ Any:
            - Regular-expression,
- And
    - After
        - ▹ Executing:
            - Statements

written

- There,
    - The
        - ▹ Program:
            - Will

start

- From
    - The:
        - ▹ Initial
            - State.
- And
    - So
        - ▹ If:
            - There

is

- No:
    - Default

            ▹ Area,

- And
    - The
        - ▹ Program:
            - Enters

the

- Sink
    - State,
        - ▹ Then:
            - It

will

- Start
    - From
        - ▹ The:
            - Initial-state.
- And
    - If
        - ▹ We
            - Write:

$$\text{someLabel.somePrivateField}: \quad \text{for}(\ldots)\{\ldots\} \tag{86}$$

- Then:

$$\text{somePrivateField}$$

will

- Be
    - Added

            ▷ Into:
                - The class

as

- A private
    ○ Field,
        ▷ And:
            - When

we

- Leave
    ○ Statement 86,
        ▷ The:
            - State

of

- The
    ○ Construct
        ▷ Will:
            - Be saved.
- And
    ○ When
        ▷ We:
            - Re-enter statement 86

the

- Program
    ○ Will
        ▷ Continue:
            - From

the

- Old
    - State.
        - ▷ But:
            - If

we

- Execute:

somePrivateField.init;

- And
    - Enter
        - ▷ Statement 86:
            - The program

will

- Start
    - From
        - ▷ The:
            - Initial-state.
- And
    - So
        - ▷ Each:
            - Time

we

- Enter:

$$\text{for}(8, \quad 80; \quad \ldots; \quad \ldots)\{\ldots\}$$

the

- Program
    - Will
        - ▹ Start:
            - – From

the

- Initial
    - State.
        - ▹ And:
            - – If

two

- Of
    - These
        - ▹ Constructs:
            - – Are nested,

the

- String
    - Of
        - ▹ The:
            - – Outer-one

will

- Not
    - Be
        - ▹ Used:
            - – In

the

- Inner
  - One.
    - ▷ And
      - If:

$$\text{for.short}(\ldots)\{\ldots\}$$

then

- Shortest
  - Match
    - ▷ Criterion:
      - Will

be

- Used.
  - And
    - ▷ If
      - We write:

$$\text{static} \quad \text{sym0}, \quad \text{sym1}, \quad \text{sym2}, \quad \text{sym3}, \quad \text{sym4}, \quad \text{sym5}; \tag{87}$$

in

- The
  - Class
    - ▷ Body,
      - Then:

$$\begin{array}{ccc} \text{sym0}, & \text{sym1}, & \text{sym2}, \\ \text{sym3}, & \text{sym4}, & \text{sym5} \end{array}$$

will

- Be
  - The
    - Symbols:
      - Of

the

- Grammar,
  - And
    - Only:
      - They

will

- Be
  - The
    - Symbols:
      - Of

the

- Grammar.
  - And
    - After:
      - Writing statement 87,

the

- Ids
  - Of:

sym0,	sym1,	sym2,
sym3,	sym4,	sym5

will

- Be:

0,	1,	2,
3,	4,	5

respectively.

- And
  - So
    - ▹ There:
      - Can

be

- Only
  - One
    - ▹ Statement:
      - Like statement 87,
- And
  - It
    - ▹ Can:
      - Only

be

- Written
  - In
    - ▹ The:
      - Body

of

- The
    - Class.
        - And:
            - After

writing

- Statement 87:

$$sym0$$

can

- Be
    - Used
        - Instead
            - Of:

$$0,$$

- And
    - So:
        - Forth.
- And so
    - After
        - Writing:
            - Statement 87,

the

- Stream
    - Given
        - To:

– The parser

should

- Be
  - Of
    - ▹ Type:

[int].

- But
  - If:

```
class  ClassOne{
    public   int              tokenId;
    public  ClassOne(){}
    ⋮
}
class  ClassTwo{
    (ClassOne.tokenId)static    sym0,    sym1,    ...;
    public  ClassTwo(){}
    ⋮
}
```

the

- Stream
    - Given
        - ▹ To:
            - The parser

should

- Be
    - Of
        - ▹ Type:

[ClassOne],

- And:

tokenId

of

- All
    - Objects
        - ▹ In:
            - That list

will

- Hold
    - The
        - ▹ Symbol:
            - Id.
- Note that:

tokenId

- Of:

ClassOne

- Maybe:

public *or* protected *or* private.

- But
  - If
    - It
      - Is:

protected *or* private,

then

- It
  - Should
    - Be:
      - Visible

in

- The
  - Environment.
    - And
      - Since:

```
class   SuperClass{

    public     int                         tokenId;

    (tokenId)static                        sym0,     sym1,     sym2;

    //  Or:   (this.tokenId)static         sym0,     sym1,     sym2;

    public   SuperClass(){}

    ⋮

}

class   SubClass   extends   SuperClass{

    (tokenId)static                        sym0,     sym1,     sym2;

    public   SubClass(){}

    ⋮

}
```

is

- Like:

$$\{\ a\ \} == \{\ a,\ a\ \},$$

we see that,

- There
    - Will
        - Be:
            - No change

in

- The
    - List
        - Of:
            - Symbols,
- And
    - Also
        - In:
            - Their ids.
- But
    - If:

```
class  SuperClass{
    public   int      tokenId;
    (tokenId)static    sym0,   sym1,   sym2;
    public  SuperClass(){}
    ⋮
}
class  SubClass  extends  SuperClass{
    (tokenId)static    sym3,   sym4;
    public  SubClass(){}
    ⋮
}
```

- Then:

sym3 *and* sym4

will

- Added
  - Into
    - The:
      - List

of

- Symbols

- ○ Inherited
  - ▷ From:

SuperClass

with

- Ids:

3 *and* 4

respectively.

- Note that:

```
class  SuperClass{
    public    int        tokenId;
    (tokenId)static      sym0,    . . . ;
    public  SuperClass(){}
    ⋮
}
class  SubClass  extends  SuperClass{
    public  SubClass(){}
    static                 sym10,    . . . ;
    ⋮
}
```

should

- Be
  - Rewritten
    - ▹ As:

```
class SuperClass{
    public int tokenId;
    (tokenId)static sym0, ...;
    public SuperClass(){}
    ⋮
}
class SubClass extends SuperClass{
    public SubClass(){}
    (tokenId)static sym10, ...;
    ⋮
}
```

- And
  - Since
    - ▹ We:
      - Can

do

- Everything
  - If
    - Their:
      - Ids

are

- Of
  - Type:

$$\text{int},$$

we

- Do
  - Not
    - Allow:
      - Them

to

- Be
  - Of
    - Type:

$$\text{float}, \quad \text{string}, \quad \ldots.$$

- And
  - If:

$$\text{static} \quad \text{sym0}, \quad (1)\text{sym1}, \quad \text{sym2}, \quad \text{sym3}, \quad \text{sym4}; \qquad (88)$$

- Then:

$$\text{sym1}$$

will

- Be
  - The:
    - Start
      - Symbol,
- And
  - Its
    - Id:
      - Can

be

- Calculated
  - As:
    - Before.
- And
  - Only
    - One:
      - Symbol

can

- Be
  - Type
    - Casted
      - With:

$$(1).$$

- Assume
  - That
    - Statement 88:

– Has

been

- Written
    - In:

SuperClass.

- Then
    - If
        - ▹ We
            - – Write:

static (1)sym1;

- In:

SubClass,

there

- Will
    - Be
        - ▹ No:
            - – Change

in

- The:
    - Start
        - ▹ Symbol.
- But
    - If:

static (1)sym2;

- Then:

sym2;

will

- Be
  - The
    - ▷ Start-symbol
      - In:

SubClass,

- And
  - There
    - ▷ Will:
      - Be

no

- Change
  - In:
    - ▷ Symbol
      - Ids.
- And
  - If:

static (1)sym5;

- Then:

sym5

will

- Be
    - Added
        - ▷ Into:
            - The list

of

- Symbols
    - As
        - ▷ The:
            - Start-symbol.
- And
    - If:

static sym0, (1)sym1, sym2, (0)symForNullString, sym3;

- Then:

symForNullString

will

- Be
    - Used
        - ▷ To:
            - Represent

the

- Empty
    - String,
        - ▷ And:
            - Its id

will

- Be:

$$3.$$

- And
    - We
        - ▹ Can:
            - Give

a

- Similar
    - Description
        - ▹ For:
            - Empty-string.
- And:

$$-1$$

will

- Be
    - Used
        - ▹ To:
            - Represent

the

- End
    - Marker.
        - ▹ And:
            - If

we

- Write:

static someRule = { sym0 : sym1 sym2 sym3 };

in

- The
  - Class
    - ▷ Body,
      - Then:

sym0 : sym1 sym2 sym3

will

- Be
  - Added
    - ▷ Into:
      - The list

of

- Rules
  - Of
    - ▷ The:
      - Grammar

with

- Name:

someRule.

- And
  - So:
    - ▷ Rule

            - Values

should

- Be

    ◦ Given

        ▹ When:

            - They

are

- Declared.

    ◦ And

        ▹ These:

            - Rules

can

- Only

    ◦ Be

        ▹ Written:

            - In

the

- Class

    ◦ Body.

        ▹ And:

            - These rules

can

- Be

    ◦ Overridden

        ▹ In:

      - Subclasses.

- And
  - Symbols
    - Written:
      - At

the

- Left
  - Hand
    - Side:
      - Of

all

- Rules
  - Will be
    - The:
      - Non-terminals.
- And
  - We
    - Can:
      - Define

the

- Symbols
  - In
    - The:
      - Super-class,
- And

    - The
        - Rules:
            - In

a

- Subclass.
    - But
        - The:
            - Reverse

is

- Not
    - Allowed.
        - Or:
            - The symbols

used

- In
    - The
        - Rules:
            - Should

be

- Defined
    - In
        - The:
            - Environment.
- And
    - If
        - We

        – Write:

static sym0, sym1, sym2, sym3, sym4;

- In:

SuperClass,

it

- Would
  - Mean:
    - ▹ That,

the

- Start
  - Symbol
    - ▹ Has:
      - – Not

yet

- Been
  - Given.
    - ▹ And:
      - – So

the

- Rules
  - Cannot
    - ▹ Be:
      - – Used

unless

- We
  - Give
    - ▹ The:
      - Start-symbol

in

- A subclass.
  - And
    - ▹ If:

$$\text{static} \quad \text{sym0}, \quad \text{sym1}[100], \quad \text{sym2}, \quad \text{sym3}[800], \quad \text{sym4};$$

the

- Priority
  - Of:

$$\text{sym0}, \qquad \text{sym2} \qquad \textit{and} \qquad \text{sym4}$$

will

- Be:

$$0,$$

- And
  - That
    - ▹ Of:

$$\text{sym1} \qquad \textit{and} \qquad \text{sym3}$$

will

- Be:

$$100 \qquad \textit{and} \qquad 800$$

respectively.

- And
  - If:

static rule1 = { sym2 : sym0 sym2 sym1 };

then

- The
  - Priority
    - Of:

rule1

will

- Be:

0.

- But
  - If:

static rule2[80] = { sym2 : sym0 sym2 sym1 };

then

- The
  - Priority
    - Of:

rule2

will

- Be:

80.

- And
    - So
        - All:
            - Conflicts

should

- Be
    - Resolved
        - During:
            - Compilation.
- And
    - These
        - Priorities:
            - Can

be

- Changed
    - In:
        - Subclasses.
- And
    - All
        - Symbols
            - Are:

public,

- And
    - All

- ▹ Rules
  - Are:

protected.

- And
  - So
    - ▹ We:
      - Do

not

- Tag
  - Them
    - ▹ With:

private *or* protected *or* public.

- But
  - We
    - ▹ Can
      - Write:

final static rule0 = { sym2 : sym0 sym2 sym1 };

- And:

final static sym0, . . . ;

is

- Equivalent
  - To.

static sym0, . . . ;

- Let:

$$\text{rule1}, \qquad \text{rule2}, \qquad \ldots$$

be

- The
  - Rules,
    - And
      - Let:

$$\text{k5}$$

be

- An
  - Instance
    - Of:

$$[\text{int}].$$

- The
  - Interpretation
    - Of:

```
int  i  =  for(int  j;  k5){

   case  rule1  :  statement_1;
                   continue;

   case  rule2  :  statement_2;
                   continue;

}
catch(k5[j − 1] == sym1 && k5[j] == sym2){
    ⋮

}
catch(k5[j + 1] == sym2 && k5[j] == sym1){

    ⋮

}
```

is

- That,
    - The
        - ▹ Stream:
            - Given

to

- The
    - Parser
        - ▹ Is:

k5,

- And
    - When
        - ▹ There:
            - Is

a

- Reduction
    - Using:

rule1,

the

- Corresponding
    - Statements
        - Will:
            - Be executed.
- And
    - If
        - The:
            - Reduction

was

- Done
    - While
        - Scanning,
            - Say:

“*the* 8th *token*,”

- Then:

$$j == 8.$$

- And

$$i == 1,$$

if

- Parsing
    - Was
        - Successful:

            - Even

if

- Some
    - Tokens
        - ▷ Had:
            - To

be

- Inserted,
    - And:

$$i \;==\; -1,$$

if

- Parsing
    - Was
        - ▷ Unsuccessful:
            - Even

after

- Inserting
    - Some:
        - ▷ Tokens.
- And
    - Parsing
        - ▷ Would:
            - Be unsuccessful,

if

- An
    - Exception
        - Is:
            - Uncaught.
- And
    - If
        - We
            - Execute:

$$\text{k5} = \text{k5}[\ ..\ \text{j} + 1] + \text{sym2} + \text{k5}[\text{j} + 1\ ..];$$

$$\text{j} = \text{j} - 2;$$

continue;

in

- The
    - Catch
        - Block:
            - The program

will

- Insert:

$$\text{sym2}$$

at

- The
    - Specified:
        - Location,

- And
  - Continue
    - Parsing:
      - From

the

- New
  - Position
    - Given
      - In:

j,

after

- Executing.

continue;

- And
  - So
    - If:
      - We

did

- Not
  - Execute:

continue;

the

- Construct
  - Will

▷ Return:

$$-1.$$

- And
  - Statements
    - ▷ Like:

$$k5 = k5[..j + 1] + sym2 + k5[j + 1..];$$

can

- Also
  - Be
    - ▷ Written:
      - In

the

- Main
  - Body
    - ▷ Of:
      - The construct.
- And
  - If
    - ▷ The:
      - Ids

are

- Of
  - Type:

SomeClass.tokenId,

we

- Can
  - Write:

```
SomeClass    sc    =    ...;
sc.tokenId         =    sym1;
// Code to insert: sc here.
```

- And
  - If:

$$\text{int int t} = \text{for}(\ldots)\{\ldots\}$$

- Then:

$$\text{t}[0] == 1 \qquad \textit{or} \qquad \text{t}[0] == -1$$

if

- Parsing
  - Was successful
    - ▷ Or unsuccessful:
      - Receptively,
- And:

$$\text{t}[1]$$

will

- Hold
  - The

        ▷ Number:
            – Of exceptions

that

- Where

    ○ Caught.

        ▷ And
            – If:

$$\text{someLabel.somePrivateField}: \quad \text{int} \quad i \quad = \quad \text{for}(\ldots)\{\ldots\}$$

- Then:

$$i == 0 \qquad \textit{and} \qquad \text{somePrivateField}[0] == 0,$$

if

- Parsing

    ○ Has

        ▷ Not:
            – Yet

been

- Completed,

    ○ And:

$$\text{somePrivateField}[0] == -2,$$

if

- We

    ○ Have:

        ▷ Not

yet

- Started
    - Parsing.
        - ▷ And:

$$\text{somePrivateField}[1]$$

will

- Hold
    - The
        - ▷ Number:
            - Of exceptions

that

- Where
    - Caught.
        - ▷ And
            - In:

$$int \quad i \quad = \quad for(; \quad scl)\{\ldots\}$$

we

- Cannot
    - Know
        - ▷ The:
            - Ordinal-position

of

- The
    - Token
        - ▷ That:
            - Was scanned.

- And
    - Since
        - ▹ All:
            - Conflicts

should

- Be
    - Resolved
        - ▹ During:
            - Compilation,

there

- Will
    - Be
        - ▹ No:
            - Default-area.
- And
    - If
        - ▹ Two:
            - Of

these

- Constructs
    - Are
        - ▹ Nested:
            - The rules

of

- The

        ◦ Outer
            ▹ One:
                – Will

not

- Be
    ◦ Applicable:
        ▹ In

the

- Inner-one,
    ◦ And:
        ▹ Vice
            – Versa.
- And
    ◦ Since
        ▹ Symbols
            – Are:

public,

the

- Token-stream
    ◦ Can
        ▹ Be generated:
            – Anywhere.
- But
    ◦ Since
        ▹ Rules

        – Are:

protected,

we see that,

- Parsing
    - Can
        - ▷ Only:
            - Be done

in

- The
    - Class
        - ▷ In:
            - Which

they

- Are
    - Written.
        - ▷ And:

$$\text{int} \quad i \quad = \quad \text{for.downwards}(\ldots)\{\ldots\}$$

can

- Be
    - Used
        - ▷ To:
            - Generate

a

- Top

    - ○ Down:
        - ▷ Parser.

- And
    - ○ We
        - ▷ Do:
            - The same

for

- All
    - ○ Other:
        - ▷ Grammars.

- And
    - ○ If:

```
while(...){
    ⋮
    someLabel:  while(...){
        ⋮
    }
    ⋮
}
catch(...){
    ⋮
    continue  someLabel;
    ⋮
}
```

throws

- An
  - ○ Exception,
    - ▷ Then:
      - Immediately

after

- Executing:

```
continue  someLabel;
```

the

- Program
  - Will
    - Start
      - From:

someLabel.

- But
  - If:

$$\text{while}(\ldots)\{\ \ldots\ \text{while}(\ldots)\{\ldots\}\ \ldots\ \}\text{catch}(\ldots)\{\ \ldots\ \text{continue;}\ \ \ldots\ \} \quad (89)$$

throws

- An
  - Exception,
    - Then:
      - Immediately

after

- Executing:

continue;

that

- Statement 89
  - Will
    - Be:
      - Reexecuted,
- And
  - Similarly,
    - For:

$$\text{if } (\ldots)\{\ldots\} \text{ else } \{\ldots\} \text{ catch}(\ldots)\{\ldots\}$$

- And
  - All
    - ▷ Other:
      - Compound-statements,
- And:

$$\text{try}\{\ldots\}\text{catch}()\{\ldots\}$$

can

- Be
  - Enhanced
    - ▷ To
      - Include:

```
continue;
```

- And:

```
for(...)
    for(...){
        ⋮
    }
```

can

- Be
  - Rewritten

▹ As:

$$\text{for}(\dots)(\dots)\{\dots\}$$

if

- Case
    - Statement:
        - ▹ Is

not

- Associated:
    - With
        - ▹ It.
            - Let:

$$G$$

be

- Some
    - Grammar.
        - ▹ Then:
            - We

can

- Check
    - Whether
        - ▹ The:
            - Length

of

- The:

- ○ Left
  - ▹ Hand
    - – Side

of

- All
  - ○ Rules
    - ▹ Is
      - – Equal to:

$$1.$$

- And
  - ○ So
    - ▹ We:
      - – Can

check

- Whether:

$$G$$

is

- Context
  - ○ Free
    - ▹ Or:
      - – Not.
- And
  - ○ So
    - ▹ From:
      - – This,

- And
    - Since:

"Turing-machines"

have

- More
    - Computational
        - Power
            - Than:

"context-free-grammars,"

we see that,

- We
    - Can
        - Construct:

"a machine"

such that,

- It will
    - Perform
        - Any:
            - Computation

on

- The
    - Given
        - Context-free:
            - Grammar,

- And
    - There
        - ▷ Will:
            - Be

no

- Self
    - Reference.
        - ▷ And:
            - So

we see that,

- There
    - Will
        - ▷ Exist:

“*a machine*”

such that,

- Given:

“*a context-free-grammar,*”

it

- Will
    - First
        - ▷ Check:
            - Whether

the

- Given

  - Grammar
    - Is:
      - Deterministic,

- And
  - If
    - It:
      - So,

that

- Machine
  - Will generate
    - The:
      - Parser-table,

else

- Enumerate
  - All conflicts.
    - And:
      - So

we see that,

- We
  - Need
    - Not:
      - Mention

the

- Value
  - Of:

$$k,$$

- But
  - Generate
    - ▷ General:
      - Parsers.

3 Declarative

Let:

$$\text{fointer1} \qquad \textit{and} \qquad \text{fointer2};$$

be

- Of
  - Type:

$$(\text{int}|\text{int}).$$

- Then
  - They
    - ▷ Can:
      - Point

to

- Methods
  - Like:

$$\text{int methodOne(int);} \qquad \textit{and} \qquad \text{int methodTwo(int);}$$

- And
  - If:

```
(int|int, float)            fointer3;
(int|)                      fointer4;
(|int)                      fointer5;
(int int|int)               fointer6;
(int|(int|int), float)      fointer7;
((int|int)|int)             fointer8;
((int|int)[ ][ ]|int)       fointer9;
([(int|int)]|int)           fointer10;
((int|int) (int|int)|int)   fointer11;
([(int|int)][ ][ ]|int [ ][ ])  fointer12;
```

- Then:

fointer3

can

- Point
  - To
    - Methods
      - Like:

int someMethod(int, float);

- And:

fointer4

- To:

$$\text{int}\ \ \text{someMethod}();$$

- And:

$$\text{fointer5}$$

- To:

$$\text{void}\ \ \text{someMethod}(\text{int});$$

- And
    - So:
        - Forth.
- And
    - We
        - Can
            - Write:

$$\begin{array}{llll} (\text{int}|\text{int},\ \text{int})\ [\,]\ \ \text{arr} & = & \text{new}\ (\text{int}|\text{int},\ \text{int})[10]; \\ \text{arr}[0] & = & (\text{i1},\ \text{i2})\{\ \text{return}\ \text{i1} + \text{i2};\ \}; \\ \text{int}\qquad\qquad\qquad \text{i} & = & \text{arr}[0](10,\ 20); \end{array}$$

- And.

```
class ClassOne{
    public (int|int) fointer = null;
    public ClassOne(){
        fointer = methodOne;
    }
    private int methodOne(int i){...}
}
class ClassTwo{
    public ClassOne co = ...;
    public ClassTwo(){
        co.fointer = methodTwo;
    }
    private void v(){...}
    private int methodTwo(int i){
        (|) v = ...;
        v();
        // If we write: this.v();
        // we will be refering to
        // the method: void v(); of this class.
        }
    }
```

- Let:

uc

be

- An
    - Instance
        - Of.

```
class UncompilableClass{
    public  (|)  v;
    public UncompilableClass(){}
    public void v(){}
}
```

- Then
    - We
        - Can:
            - Say

that,

- If
    - We

        ▷ Write:

uc.v();

we

- Will
    - Be
        ▷ Referring:
            - To

the

- Method,
    - And
        ▷ If:

(class.field)uc.v();

we

- Will
    - Be
        ▷ Referring:
            - To

the

- Field.
    - But
        ▷ We:
            - Avoid it.
- And
    - So:

UncompilableClass

will

- Not
  - Compile.
    - And:
      - We

say

- That:

```
(int|int)    fointer1;
```

is

- Equivalent
  - To:

```
int    fointer1(int);
```

- And:

```
(int|)    fointer4;
```

- To.

```
int    fointer4();
```

- And
  - So:

```
(int|int)    fointer1;

(int|)       fointer4;

int          i    j;
```

can

- Be
    - Rewritten
        - ▹ As.

```
int    fointer1(int),    fointer4(),    i,    j;
```

- Let.

```
int      f1(int),    f2(int),    f3(int),    f4(int),    f5(int),
         f6(boolean);

float    q(int);

int      h(int,  string),    h2(int,  int),    h3(int,  int),
         h4(int,  int);

void     v1(int),    v2(),    v3(),    v4(int),    v5(int);
```

- The
    - Interpretation
        - ▹ Of:

```
void    sm()    =    v2  >  v3;
```

- Is.

```
void    sm()    =    (){

                          v2();

                          v3();

                     };
```

- And
    - That
        - Of:

```
void sm(int) = f1 > v1;
```

- Is:

```
void sm(int) = (i){
                int j1;
                j1 = f1(i);
                v1(j1);
        };
```

- And
    - That
        - Of:

```
int sm(int) = f1 > f2;
```

- Is:

```
int sm(int) = (i){
                int j1, j2;
                j1 = f1(i);
                j2 = f2(j1);
```

```
                    return  j2;
            };
```

- But:

$$\text{int} \quad \text{sm(int)} \quad = \quad \text{f1} \;>\; \text{q;}$$

will

- Not
  - Compile,
    - Since:

```
int   sm(int)   =   (i){
                    int     j1;
                    float   j2;
                    j1  =   f1(i);
                    j2  =   q(j1);
                    return  j2;
            };
```

will

- Not
  - Compile.
    - The
      - Interpretation of:

$$\text{float} \quad \text{sm(int)} \quad = \quad \text{f1} \;>\; \text{f2} \;>\; \text{q;}$$

- Is:

```
float   sm(int)   =   (i){
                          int     j1,   j2;
                          float   j3;
                          j1   =   f1(i);
                          j2   =   f2(j1);
                          j3   =   q(j2);
                          return  j3;
                      };
```

- And
  - Similarly,
    - ▹ For.

```
int   sm(int)   =   f1  >  f2  >  f3  >  f4  >  f5;
```

- The
  - Interpretation
    - ▹ Of:

```
int  int   sm(int  int)   =   f1  f2;
```

- Is:

```
int  int   sm(int  int)   =   (t){   return  f1(t[0])  f2(t[1]);   };
```

- And
  - That

        ▹ Of:

```
void sm(int, int) = f1, f2;
```

- Is:

```
void sm(int, int) = (i1, i2){
                        int j1, j2;

                        j1 = f1(i1);
                        j2 = f2(i2);

                    };
```

- And
    - That
        ▹ Of:

```
int sm(int, int, int) = h2, f3 > h3;
```

- Is:

```
int sm(int, int, int) = (i1, i2, i3){
                            int j1, j2;

                            j1 = h2(i1, i2);
                            j2 = f3(i3);

                            return h3(j1, j2);

                        };
```

- And
    - That
        - Of:

```
int  sm(int, int, string, int)  =  f1, h, f2 > h2, f3 > h3;
```

- Is:

```
    int  sm0(int, int, int)           =  h2, f3 > h3;

    int  sm(int, int, string, int)    =  (i1, i2, s, i4){

                                             int  j1,  j2,  j3;

                                             j1  =  f1(i1);
                                             j2  =  h(i2, s);
                                             j3  =  f2(i4);

                                             return sm0(j1, j2, j3);

                                         };
```

- And
    - That
        - Of:

```
int  sm(int, int, int, string)  =  f1, f2, h > h2, f3 > h3;
```

- Is:

```
    int  sm0(int, int, int)           =  h2, f3 > h3;

    int  sm(int, int, int, string)    =  (i1, i2, i3, s){
```

```
                    int   j1,   j2,   j3;

                    j1   =   f1(i1);
                    j2   =   f2(i2);
                    j3   =   h(i3,  s);

                    return  sm0(j1,  j2,  j3);

                };
```

- And
  - That
    - ▹ Of:

```
int   sm(int,  int)   =   f1,  v2,  f3  >  f4,  f5  >  h2;
```

- Is:

```
int   sm0(int,  int)   =   f4,  f5  >  h2;

int   sm(int,  int)    =   (i1,  i2){

                    int   j1,   j2,   j3;

                    j1   =   f1(i1);
                    v2();
                    j2   =   f3(i2);

                    return  sm0(j1,  j2);

                };
```

- Or
  - When

      - ▷ The:
        - Number

of

- Parameters
  - Accepted
    - ▷ By:
      - A level

is

- Equal
  - To
    - ▷ The:
      - Number

of

- Values
  - Returned:
    - ▷ By

the

- Previous level,
  - And
    - ▷ The:
      - First variable

of

- The
  - Accepting
    - ▷ Level

- Has:

“n *parameters,*”

then

- The
  - First:

“n *values-returned*”

by

- The
  - Previous
    - ▹ Level:
      - Will

be

- Given
  - To
    - ▹ The:
      - First variable,
- And
  - So
    - ▹ Forth:
      - Until everything

has

- Been
  - Given,
    - ▹ Else
      - We use:

break.

- Exemplifying:

```
int   sm(int, int, int)   =   break, f2, break;
```

is

- Equivalent
  - To.

```
int   sm(int, int, int)   =   (i1, i2, i3){
                                      int   j2;
                                      j2   =   f2(i2);
                                      return  j2;
                              };
```

- The
  - Interpretation
    - ▷ Of:

```
void   sm(int)   =   f1, f2;
```

- Is:

```
void   sm(int)   =   (i){
                           int   j1,   j2;
                           j1   =   f1(i);
                           j2   =   f2(i);
                     };
```

- And
    - That
        - Of:

```
int int sm(int) = f1 f2;
```

- Is:

```
int int sm(int) = (i){ return f1(i) f2(i); };
```

- And
    - That
        - Of:

```
int sm(int) = f1, f2 > h2;
```

- Is:

```
int sm(int) = (i){

        int j1, j2;

        j1 = f1(i);
        j2 = f2(i);

        return h2(j1, j2);

    };
```

- And
    - That
        - Of:

```
int sm(int, int) = h2, h3 > h4;
```

- Is:

```
int   sm(int,  int)   =   (i1,  i2){
                              int   j1,   j2;

                              j1   =   h2(i1,  i2);
                              j2   =   h3(i1,  i2);

                              return  h4(j1,  j2);

                          };
```

- Or
    - When
        - ▹ The:
            - – Number

of

- Parameters
    - Accepted
        - ▹ By:
            - – A level

is

- A multiple
    - Of:
        - ▹ The values
            - – Returned

by

- The previous

        ◦ Level,
            ▹ The:
                – Operation

which

- We
    - ◦ Mentioned
        - ▹ When:
            - – The number

of

- Parameters
    - ◦ On
        - ▹ Both:
            - – Sides

are

- Equal
    - ◦ Will
        - ▹ Be:
            - – Repeated

until

- Everything
    - ◦ Have
        - ▹ Received:
            - – Their parameter,

else

- We

- Use:

break.

- Exemplifying,
    - The
        - ▹ Interpretation
            - Of:

```
void    sm(int,  int)    =    v1,  v4,  break,  v5;
```

- Is:

```
void    sm(int,  int)    =    (i1,  i2){

                                    v1(i1);
                                    v4(i2);

                                    v5(i2);

                              };
```

- The
    - Interpretation
        - ▹ Of:

```
int    i1,    i2;

int    sm(int,  int)    =    ...;

int    i3               =    i1,  i2  >  sm;
```

- Is:

```
int   i1,   i2;

int   sm(int,  int)     =    ...;

int   i3                =    sm(i1,  i2);
```

- And
    - That
        - Of:

```
int   i;

int   sm()   =   i  >  f1  >  f2;
```

- Is:

```
int   i;

int   sm()   =   (){   return  f2(f1(i));   };
```

- And
    - That
        - Of:

```
int   i;

int   sm(int)   =   i  >  f1  >  f2;
```

- Is:

```
int    i;
int    sm(int)    =    (i1){
                                 i    =    i1;
                                 return  f2(f1(i));
                            };
```

- And
  - That
    - Of:

```
int    i1,    i2;
int    sm(int)     =    f1  >  i1  >  i2  >  f2;
```

- Is:

```
int    i1,    i2;
int    sm(int)    =    (i3){
                               int    j1;
                               j1    =    f1(i3);
                               i1    =    j1;
                               i2    =    i1;
                               return  f2(i2);
```

```
                            };
```

- And
    - That
        - Of:

```
int   i;

int   sm(int)   =   f1  >  i;
```

- Is:

```
int   i;

int   sm(int)   =   (i1){

                         int   j1;

                         j1   =   f1(i1);

                         i    =   j1;

                         return  i;

                    };
```

- And
    - That
        - Of:

```
          int   sm(int)   =   f1  >  8  >  f2;
```

- Is:

```
int   sm(int)   =   (i){
                       int   j1,   j2;
                       j1   =   f1(i);
                       j2   =   f2(8);
                       return  j2;
                    };
```

- And
    - That
        - ▷ Of:

```
          int   sm(int)   =   f1  >  8;
```

- Is:

```
int   sm(int)   =   (i){
                       int   j1;
                       j1   =   f1(i);
                       return  8;
                    };
```

- And

- That
    - Of:

```
int   i;
int   sm()   =   i > 8;
```

- Is:

```
int   i;
int   sm()   =   (){   return 8;   };
```

- And
  - That
    - Of:

```
int   i;
int   sm()  =   i;
sm          =   0;
```

- Is:

```
int   i;
int   sm()  =  (){   return i;   };
sm          =  (){   return 0;   };
```

- And
    - That
        - Of:

$$\text{void} \quad \text{sm(int)} \quad = \quad 8;$$

- Is:

$$\text{void} \quad \text{sm(int)} \quad = \quad \text{(i)}\{\};$$

- Or
    - If
        - The:
            - Next-level

cannot

- Receive
    - Any:
        - Parameter,

then

- All
    - Values
        - Returned:
            - By

the

- Previous
    - Level
        - Will:
            - Be discarded,

else

- If
    - Parameters
        - To:
            - All variables

in

- The
    - Next
        - Level:
            - Cannot

be

- Given,
    - Then
        - There:
            - Will

be

- An
    - Error.
        - And
            - So:

$$\text{void} \quad \text{sm()} \quad = \quad \text{f1;}$$

will

- Produce
    - An:
        - Error,

since

- The
    - Parameters
        - ▹ Of:

f1

could

- Not
    - Be:
        - ▹ Given.
- And
    - Similarly:

```
int     i,   j;

void    sm   =   i > j > f6;
```

will

- Not
    - Compile.
        - ▹ But:

```
int     i,   j;

void    sm   =   (i > j) > f6;
```

is

- Equivalent
  - To.

```
int     i,   j;

void    sm()   =   (){

                        boolean   b;

                        int       i1;

                        b    =    (i  >  j);

                        i1   =    f6(b);

                   };
```

- And
  - In:

```
int    sm(int,  int)    =    f1,  f2  >  ...;
```

we

- Can
  - Use:

sm[0][0] *and* sm[1][0]

to

- Reference
  - The:
    - First

      - Value-returned

by

- The
    - Zeroth
        - ▷ And:
            - First-levels,
- And:

$$\text{sm}[0][1] \qquad \textit{and} \qquad \text{sm}[1][1]$$

to

- Reference
    - The:
        - ▷ Second
            - Value-returned

by

- The
    - Zeroth
        - ▷ And:
            - First-levels,
- And
    - So:
        - ▷ Forth.
- And so
    - The
        - ▷ Interpretation
            - Of:

int sm(int) = f1 > f2 > f3 > f4 > sm[1][0] + sm[4][0] > f5;

- Is:

```
int sm(int) = (i){
        int j1, j2, j3, j4, j5;
        j1 = f1(i);
        j2 = f2(j1);
        j3 = f3(j2);
        j4 = f4(j3);
        j5 = f5(j1 + j4);
        return j5;
    };
```

- And
    - That
        - Of:

int sm(int, string) = (h2 : h3 ? sm[0][0] > 8 || sm[0][1] != "a");

- Is:

```
int   sm(int, string)   =   (i, s){
                                   if (i > 8 || s != "a")
                                      return h2(i, s);
                                   else
                                      return h3(i, s);
                          };
```

- And
  - That
    - ▹ Of:

```
int   sm(int)   =   (f1 : sm[0][0] ? bool-Exp);
```

- Is:

```
int   sm(int)   =   (i){
                           if (bool-Exp)
                              return f1(i);
                           else
                              return i;
                     };
```

- And
  - In:

$$\text{int} \quad \text{sm(int)} \quad = \quad \text{f1} \; > \; (\text{f2} \; : \; \text{f3} \; ? \; \text{bool-Exp}) \; > \; \text{f4};$$

we

- Can
  - Use:

$$\text{sm}[2,\;1][0] \qquad \textit{and} \qquad \text{sm}[2][0]$$

to

- Reference
  - The
    - ▷ Value
      - Returned by:

$$\text{f2} \qquad \textit{and} \qquad (\text{f2} \; : \; \text{f3} \; ? \; \text{bool-Exp})$$

respectively.

- And
  - We
    - ▷ Can
      - Write:

$$\text{int} \quad \text{sm(int)} \quad = \quad (1 \;:\; \text{sm}[0][0] \; - \; 1 \; > \; \text{sm} \; ? \; \text{bool-Exp}); \tag{90}$$

- But
  - Not:

$$\text{int} \quad \text{sm(int)} \quad = \quad \text{f1} \; > \; (\text{f2} \; : \; \text{f3} \; ? \; \text{bool-Exp}) \; > \; \text{sm}[2,\;1][0] \; > \; \text{f4};$$

- And
  - We
    - ▷ Say:
      - That,

If

- The
    - Width
        - Of:
            - The statement

is

- Equal
    - To:

1,

- Then:

<fointer-name>[0][0], <fointer-name>[0][1], . . .

can

- Be
    - Rewritten
        - As:

<fointer-name>[0], <fointer-name>[1], ...

- And
    - So
        - Statement 90:
            - Can

be

- Rewritten
    - As:

$$\text{int} \quad \text{sm(int)} \quad = \quad (1 \;:\; \text{sm}[0] \;-\; 1 \;>\; \text{sm} \;?\; \text{bool-Exp});$$

- But:

$$\text{void} \quad \text{sm(int,} \;\; \text{string)} \quad = \quad \text{sm}[0][0] \;+\; 1, \;\; \text{sm}[0][1] \;+\; \texttt{"a"} \;>\; \text{h2};$$

cannot

- Be
  - Rewritten.
    - And:

$$\text{f1} \quad >= \quad \text{f2};$$

is

- Equivalent
  - To.

$$\text{f1} \quad = \quad \text{f1} \;>\; \text{f2};$$

- Let:

$$(\text{int}|\text{int}) \quad \text{fointerReturner}((\text{int}|\text{int}));$$

be

- Some
  - Method,
    - And
      - Let:

$$\text{k23} \qquad \textit{and} \qquad \text{k24}$$

be

- Instances
  - Of:

[(int|int)].

- Then:

“> f1”

will

- Be
  - Applied:
    - To

all

- Elements
  - Of:

k23[ ]

- In:

```
k23          =    fointerReturner(k24[ ]);
[int]   k5   =    k23[ ](8);
k23[ ]       >=   f1;
```

- And
  - We
    - Can:
      - Give

a

- Description

- For:

$$>=$$

like

- That
    - Which
        - ▷ We
            - Did for:

$=,\qquad +=,\qquad \&=,\qquad |=\qquad \textit{and}\qquad \%=$

in

- Sub sections 1.1 and 1.2.
    - And
        - ▷ We
            - Can write:

```
class  ClassFour{
    public   int    i;
    public  ClassFour(){
        void          v()    =     (int)i  >  i  >  (int)i;
        (int|int)     f1     =     (i){...};
        [(int|int)]   k23    +=    f1;
        k23                  +=    (i){...};
        k23                  %=    f1;
        }
        private int i(int i){...}
    }
```

- But
  - Not:

```
k23 %= (i){...};
```

- And:

```
int f1(int) = (i){
        if (i < 2)
            return i * do(i − 1);
        return 1;
    };
```

can

- Be
  - Used
    - For:
      - Recursion.
- And
  - If:

```
int     sf()         =   (){   return.outer  10;   };
int     sf2((int|))  =   (f){
                               int  i  =  f();
                               return  8;
                         };
string  sf3((int|))  =   (f){
                               int  i  =  f();
                               return  "abc";
                         };
int     i            =   sf2(sf);
string  s            =   sf3(sf);
```

- Then:

sf2

will

- Return:

10

immediately

- After
  - Executing:

int i = f();

- And:

$$\text{sf3}$$

will

- Return:

$$\text{“}\textit{default-value}\text{”}$$

instead

- Of:

$$\texttt{"abc"},$$

- And
    - Similarly,
        - ▷ In:
            - Methods.
- Let:

$$\text{final} \quad \text{final} \quad \text{int} \quad \text{ff1}(\text{int}) \quad = \quad (\text{i})\{\ldots\};$$

$$\text{final} \quad \text{final} \quad \text{int} \quad \text{ff2}(\text{int}) \quad = \quad (\text{i})\{\ldots\};$$

- And
    - Let.

```
interface InterfaceOne{
    public int methodOne(int i);
    public char methodTwo(int i);
}

interface InterfaceTwo{

    public int methodOne(int i);

    public int methodTwo(int i);

}
```

- Then
    - In:

InterfaceOne obj = (c){ return 'a'; }, (i){ return 10; };

the

- Compiler
    - Can
        - ▹ Consider:

(i){ return 10; }

as

- An
    - Implementation
        - ▹ Of:

public int methodOne(int i);

- And:

(c){ return 'a'; }

as

- An
    - Implementation
        - ▷ Of:

public char methodTwo(int i);

- And
    - Instantiate:

obj.

- And
    - We
        - ▷ Can
            - Write:

```
InterfaceTwo   obj   =   (InterfaceTwo.methodTwo)ff2,
                         (InterfaceTwo.methodOne)ff1;
obj                  =   ff2,   (InterfaceTwo.methodOne)ff2;
```

- And
    - Only:
        - ▷ Final final
            - Variables

can

- Be
    - Used
        - ▷ For:

        - This purpose

for

- The
    - Sake
        - Of:
            - Avoiding exceptions.
- And:

$$(\text{int}|\text{int}) \quad \text{f1};$$

is

- Equivalent
    - To:

$$(\text{int}|\text{int}) \quad \text{f1} \quad = \quad \text{null};$$

- And:

$$\begin{array}{llll}
\text{int} \ (\text{int}|\text{int}) \ \text{int} & \text{t13} & = & 10 \ (\text{i})\{\ldots\} \ 20; \\
\text{int} & \text{i} & = & 0, \quad \text{j} \quad = \quad 1; \\
(\text{int}|\text{int}) & \text{sm} & = & \text{f1} > \text{f2} > \text{f3} > \text{sm}[\text{i}][\text{j}] > \text{f4}; \\
(\text{int}|) \ [\,] & \text{arr} & = & (\quad ()\{\ldots\}, \quad ()\{\ldots\} \quad);
\end{array}$$

should

- Be
    - Rewritten
        - As:

```
(int|int)              f9    =   (i){...};
int  (int|int)  int    t13   =   10  f9  20;
(int|int)              sm    =   f1 >  f2  >  f3  >  sm[0][1]  >  f4;
(int|)                 f10   =   (){...},    f11    =    (){...};
(int|)  [ ]            arr   =   (    f10,    f11    );
//  And  similarly,  for  trees  and  lists.
```

- Or
    - Literals
        - Of:
            - These variables

cannot

- Be
    - Written
        - In:
            - Tuples

with

- More
    - Than
        - One:
            - Location.
- And
    - We
        - Can

- Write:

$$\text{if} \quad (\text{f1} \quad == \quad \text{f2}) \quad \text{f3} \quad = \quad \text{f4};$$

- But
  - Not:

$$\text{if} \;\; (\text{f1} \;\; == \;\; (\text{i})\{\dots\})\{\dots\}$$

- And
  - If:

$$(+)(\text{int}|\text{int}) \quad \text{fointerToFointer};$$

- Then:

$$\text{fointerToFointer};$$

can

- Point
  - To
    - The:
      - Address

stored

- In
  - An
    - Instance
      - Of:

$$(\text{int}|\text{int}).$$

- And
  - Similarly,

        ▷ For:

```
(++)(int|int)    fointerToFointerToFointer;
```

- And
    - Values
        - ▷ Of:
            - These variables

cannot

- Be
    - Committed.
        - ▷ And
            - So:

```
static class SomeSchema{
    public    native    [(int|int)]    k23;
    ⋮
}
```

will

- Not
    - Compile.
        - ▷ Let:

```
int  i,  j,  w  =  (){  return i + j;  },  i2;
```

- And

- We
    - Execute:

```
i          =  100;
j          =  200;
int   m1   =  w;
j          =  300;
int   m2   =  w;
```

- And
    - When:

```
int   m1   =  w;
```

is

- Executed,
    - First:

w

will

- Be
    - Evaluated:
        - Using

the

- Present
    - Values

            ▷ In:

$$i \quad \textit{and} \quad j,$$

- And
    - Then
        - ▷ The:
            - Result

will

- Be
    - Given
        - ▷ To:

$$m1.$$

- And
    - Similarly,
        - ▷ For.

$$\text{int} \quad m2 \quad = \quad w;$$

- And
    - So
        - ▷ After:
            - Executing

the

- Above
    - Code
        - ▷ Fragment:

$$m1 \ == \ 300 \qquad \textit{and} \qquad m2 \ == \ 400.$$

- And
    - So
        - If:

$$\text{void voidReturner(int);} \tag{91}$$

is

- Some
    - Method,
        - And
            - We execute:

$$\text{voidReturner(w);}$$

- Then:

$$w$$

be

- First
    - Evaluated,
        - And:
            - The result

will

- Be
    - Given
        - To:
            - Method 91.
- And
    - We
        - Can
            - Write:

```
int     w     =   (){
                        for(int   i   =   2;   i< 8;   i++)
                             j   =   j * i   +   20;
                             // j was defined earlier.
                        return j;
                  };
int     i,   w2    =   (){   return w * 2;   },   i2;
void    stm        =   (){   i   =   w2;   };
stm;
(int|)  f12        =   (){   return 8;   };
(int|)  w3         =   (){   return f12;   };
(int|)  f13        =   w3;
```

- And
  - The
    - Value:
      - Of

these

- Expressions
  - Cannot
    - Be:
      - Changed

after

- Initialization.
    - And
        - ▷ We:
            - Do

not

- Allow
    - Recursion
        - ▷ In:
            - Them.
- Or
    - If
        - ▷ We
            - Write:

$$<\text{variable-name}> \quad = \quad \text{do}();$$

we

- Will
    - Be
        - ▷ Referring:
            - To

the

- Method
    - In
        - ▷ Which:
            - That expression

is

- Written.
    - But
        - ▹ They:
            - Can

have

- Side
    - Effects.
        - ▹ And
            - If:

$$\text{void} \quad \text{commit} \quad = \quad ()\{ \quad \ldots; \quad \text{commit}; \quad \ldots; \quad \}; \tag{92}$$

then

- Database
    - Commit
        - ▹ Statement:
            - Will

be

- Overriden.
    - And
        - ▹ So:

$$\text{commit};$$

inside

- The
    - Block
        - ▹ Of:

    – Statement 92

means

- Database
  - Commit:
    - ▷ Statement.
- And
  - Similarly,
    - ▷ For:

```
void rollback = (){ ...; rollback; ...; };
```

- And:

```
int w = {
            for(int i = 2; i< 8; i++)
                j = j * i + 20;
                // j was defined earlier.
            return j;
        };
```

is

- Like:

```
int i1 = ..., i2 = ...;
int i3 = i1 * i2;
```

- And
    - Similarly,
        - For:

trees, lists and arrays.

- Let:

```
class ClassOne{
    private     int   a;
    protected   int   b;
    public      int   c;
    public ClassOne(){...}
    private int methodA(int i){
        a  +=  i + 1;
        return a;
    }
    protected int methodB(int i){
        b  +=  methodA(i) + 1;
        return b;
    }
    public int methodC(int i){
        c   +=   methodB(i)  +  1;
        return  c;
    }
}
```

- And
  - Let:

```
class  ClassTwo{
    private       int    a;
    protected     int    bb;
    public        int    cc;
    public  ClassTwo(){...}
    private  int  methodA(int  i){
        a   +=   i  +  2;
        return  a;
    }
    protected  int  methodBB(int  i){
        bb   +=   methodA(i)  +  2;
        return  bb;
    }
```

```
        public int methodCC(int i){
            cc += methodBB(i) + 2;
            return cc;
        }
    }
```

- And
  - Let:

```
    class ClassThree{
        private     int   a;
        protected   int   bb;
        public      int   cc;
        public ClassThree(){...}
        private int methodA(int i){
            a += i + 3;
            return a;
        }
        protected int methodBB(int i){
            bb += methodA(i) + 3;
        return bb;
    }
    public int methodCC(int i){
        cc += methodBB(i) + 3;
        return cc;
    }
}
```

- And
  - Let.

```
class ClassFour{
    public ClassFour(){...}
    public int methodCC(int i){...}
    public int methodCC(int i, int j){...}
}
```

- Then
  - For:

```
    class MixClassOne = ClassOne, ClassTwo;
```

the

- Compiler

- Will
  - Generate.

```
class MixClassOne{
    private     int   a;
    protected   int   b;
    public      int   c;
    private     int   newNameInsteadOfClassTwoDota;
    protected   int   bb;
    public      int   cc;
    public MixClassOne(){
        <ClassOne's-default-constructor-copied-here>
        <ClassTwo's-default-constructor-copied-here>
    }
    private int methodA(int i){
        a += i + 1;
        return a;
    }
    protected int methodB(int i){
        b += methodA(i) + 1;
        return b;
    }
    public int methodC(int i){
        c += methodB(i) + 1;
        return c;
    }
    private int newNameInsteadOfmethodA(int i){
        newNameInsteadOfClassTwoDota += i + 2;
        return newNameInsteadOfClassTwoDota;
    }
    protected int methodBB(int i){
        bb += newNameInsteadOfmethodA(i) + 2;
        return bb;
    }
    public int methodCC(int i){
        cc += methodBB(i) + 2;
        return cc;
    }
}
```

- Note that:

MixClassOne

is

- Not
  - A subtype
    - Of:

ClassOne *or* ClassTwo

since

- The
  - Contents
    - Of:

ClassOne *and* ClassTwo

are

- Copied
  - As
    - Such
      - Into:

MixClassOne

after

- Resolving
  - Conflicts.
    - And
      - So:

class MixClassTwo = ClassThree, ClassTwo;

will

- Not
  - Compile,
    - ▹ Since:
      - – There

are

- Conflicts.
  - But
    - ▹ For:

```
class   MixClassTwo   =   ClassThree,   ClassTwo

                 :     ClassTwo.bb             is   newbb,

                       ClassThree.cc           is   newcc,

                       ClassThree.methodBB     is   newMethodBB,

                       ClassThree.methodCC     is   newMethodCC;
```

the

- Compiler
  - Will
    - ▹ Generate:

```
class  MixClassTwo{

    private      int     a;
```

```
protected    int    bb;
public       int    newcc;
private      int    newa;
protected    int    newbb;
public       int    cc;
public  MixClassTwo(){
    <ClassThree's-default-constructor-copied-here>
    <ClassTwo's-default-constructor-copied-here>
}
private  int  methodA(int  i){
    a   +=   i + 3;
    return  a;
}
protected  int  newMethodBB(int  i){
    bb   +=   methodA(i)  +  3;
    return  bb;
}
public  int  newMethodCC(int  i){
    newcc   +=   newMethodBB(i)  +  3;
```

```
        return newcc;
    }
    private int newMethodA(int i){
        newa += i + 2;
        return newa;
    }
    protected int methodBB(int i){
        newbb += newMethodA(i) + 2;
        return newbb;
    }
    public int methodCC(int i){
        cc += methodBB(i) + 2;
        return cc;
    }
}
```

- And
  - For:

```
class   MixClassThree   =   ClassThree,    (public  int)dd,
                              (public  (int|int))methodCC
      :   ClassThree.bb           is   (private)bb,
          ClassThree.cc           is   (protected)cc,
          ClassThree.methodBB     is   (private)methodBB,
          ClassThree.methodCC     is   (private)newName,
          methodCC                is   ClassThree.methodBB
                                         >  dd
                                           >  ClassThree.methodCC;
// Or  we  can  write:
// methodCC   is   methodBB  >  dd  >  newName.
```

the

- Compiler
    - Will
        - Generate.

```
class  MixClassThree{
    private      int    a;
    private      int    bb;
    protected    int    cc;
```

```
        public          int       dd;

        public  MixClassThree(){

            <ClassThree's-default-constructor-copied-here>

        }

        private  int  methodA(int  i){...}

        private  int  methodBB(int  i){...}

        private  int  newName(int  i){...}

        public  int  methodCC(int  i){

            return  i  >  methodBB  >  dd  >  newName;

        }

    }
```

- Note that,
    - Private
        - ▷ Members:
            - Cannot

be

- Renamed.
    - And
        - ▷ Protected:
            - Members

can

- Be
  - Converted
    - To:

private,

- And
  - Public
    - Members
      - To:

protected *or* private.

- But
  - The
    - Reverse:
      - Is

not

- Permitted.
  - And
    - In:

```
class MixClassFour = (private)ClassOne,
                     (protected)ClassTwo,
                     ClassThree : ...;
```

all

- Protected

- And
    - Public
        - Members of:

ClassOne

will

- Be
    - Converted
        - To:

private,

- And
    - All
        - Public-members
            - Of:

ClassTwo

will

- Be
    - Converted
        - To:

protected.

- And
    - For:

$$\begin{aligned}
\text{class}\quad \text{MixClassFive}\quad = \quad & (\text{private}\ \ \text{int})\text{dd},\\
& (\text{public}\ \ (\text{int}|\text{int}))\text{methodA},\\
& (\text{public}\ \ (\text{int}|\text{int},\ \ \text{int}))\text{methodA}
\end{aligned}$$

$$\begin{array}{llll}
: & ((\text{int}|\text{int}))\text{methodA} & \text{is} & \text{ClassThree.methodCC},\\
 & \text{methodA} & \text{is} & ((\text{int}|\text{int},\ \ \text{int}))\text{ClassFour.methodCC}\\
 & & & \qquad > \quad \text{dd};
\end{array}$$

the

- Compiler
  - Will
    - Generate:

```
class  MixClassFive{
    private    int    a;
    private    int    bb;
    private    int    cc;
    private    int    dd;
    public  MixClassFive(){
        <ClassThree's-default-constructor-copied-here>
        <ClassFour's-default-constructor-copied-here>
    }
    private  int  newNameForMethodA(int  i){
        <body-of-ClassThree's-methodA-copied-here>
    }
    private  int  methodBB(int  i){
        <body-of-ClassThree's-methodBB-copied-here>
    }
    private  int  methodCC(int  i){
        <body-of-ClassThree's-methodCC-copied-here>
    }
    private  int  newNameForMethodCC(int  i){
        <body-of-ClassFour's-methodCC-copied-here>
    }
    private int methodCC(int i, int j){
        <body-of-ClassFour's-methodCC-copied-here>
    }
    public int methodA(int i){
        return newNameForMethodCC(i);
    }
    public int methodA(int i, int j){
        return i, j > methodCC > dd;
    }
}
```

- And
  - Only
    - ▹ Public:
      - Methods

can

- Be
  - Imported
    - ▹ This:
      - Way.

- And
    - If:

```
interface SomeInterface{
    public void methodDD(int i);
}
```

- Then:

```
class InnerClass implements SomeInterface{
    public              (enum)int       z;
    public              [int]           k5;
    public      static  int [ ]         arr;
    // Body of: ClassThree copied here.
    static{
        initS();
    }
    public InnerClass(){
        <ClassThree's-Default-Constructor-Copied-here>
        constructor();
    }
    private int f1(int i){
        cc      =    someMethod(0);
        return cc;
    }
    private void f2(){
        z[0]    =    methodCC(0);
        k5      =    10, 20;
    }
    private static void f3(){
        arr     =    ( 0 );
    }
    private void constructor(){ f2(); }
    private static void initS(){ f3(); }
    public int methodDD(int i){ return f1(i); }
}
```

will

- Be
  - Generated
    - ▹ Inside:

```
class   SomeClass{

    public                final    final    int       f1(int)    =    (i){

        class.cc               =    someMethod(0);

        return   class.cc;

    };

    public                final    final    void      f2()       =    (){

        class.z[0]             =    class.methodCC(0);

        class.k5               =    10,   20;

    };

    public      static    final    final    void      f3()       =    (){

        class.arr              =    (   0   );

    };

    class     InnerClass       =    SomeInterface,

                                    ClassThree,
                    (public (enum)int)z,
                    (public [int])k5,
                    (public static int [ ])arr;
                    (private.init (|))constructor,
                    (private static.init (|))initS,
                    (public (int|int))methodDD
            : constructor   is   f2,
              initS         is   f3,
              methodDD      is   f1;
    // And: (private static.init (|)) and (private.init static (|))
    // are equivalent.
    public SomeClass(){...}
    public int someMethod(int i){...}
}
```

- And:

```
    public        final   final   int    f1(int)   =   (i){
        class.cc    =    someMethod(0);
        return class.cc;
```

```
};
public          final   final   void   f2()    =   (){
    class.z[0]   =   class.methodCC(0);
    class.k5     =   10,  20;
};
public  static  final   final   void   f3()    =   (){
    class.arr    =   (  0  );
};
```

will

- Be
  - Converted
    - ▷ To:

```
public          final   final   int    f1(int)  =   (i){
    return  <default-value>;
};
public          final   final   void   f2()     =   (){};
public  static  final   final   void   f3()     =   (){};
```

- And
  - For:

```
final   final   int   f4(int)   =   (i){
        class.a    =   i;
        class.bb   =   i;
        class.dd   =   i;
        return  class.a  +  class.bb  +  class.dd;
};
class   MixClassSix            =   ClassThree
                                   (public  (int|int))methodDD
                          :   methodDD    is    f4;
```

the

- Compiler
  - Will
    - Generate:

```
class  MixClassSix{
    private      int    a;
    protected    int    bb;
    public       int    cc;
    public  MixClassSix(){
        <ClassThree's-default-constructor-copied-here>
    }
    private int methodA(int i){...}
    protected int methodBB(int i){...}
    public int methodCC(int i){...}
    private int f4(int i){
        bb = i;
        return <default-value> + bb + <default-value>;
    }
    public int methodDD(int i){
        return i > f4;
    }
}
```

- Since:

a

is

- Private
  - In:

ClassThree.

- And
  - Only:

        - ▹ Final final
            - Variables

can

- Be
    - Used
        - ▹ For:
            - This purpose.
- And
    - So
        - ▹ In:
            - General,

the

- Part
    - Between:

‘=’ *and* ‘:’

can

- Contain:

classes *and* *interfaces*

- And
    - Methods
        - ▹ That:
            - Are imported,
- And
    - In

      - ▹ The:
        - Part

after

- Colon:

protected-members *and* *public-members*

of

- Imported
  - Classes
    - ▹ Can:
      - Be renamed,
- And
  - The
    - ▹ Methods:
      - Of

the

- Class
  - Can
    - ▹ Be:
      - Constructed.
- And
  - Final
    - ▹ And partial:
      - Things

cannot

- Be

- Imported
  - ▷ This:
    - Way.

- And
  - There
    - ▷ Will:
      - Be

an

- Error,
  - If
    - ▷ The:
      - Default-constructor

of

- An
  - Imported
    - ▷ Class:
      - Is

not

- Visible
  - In
    - ▷ The:
      - Environment.
- And:

ClassThree *and* ClassTwo

will

- Be
    - Joined
        - Together
            - At:

bb

- In:

```
class   MixClassSeven   =   ClassThree,   ClassTwo
             :      ClassThree.bb          is   ClassTwo.bb,
                    ClassThree.cc          is   . . . ,
                    ClassThree.methodBB    is   . . . ,
                    ClassThree.methodCC    is   . . . ;
```

- And
    - Even
        - Though:

```
class   MixClassEight   =   ClassThree,
                            ClassTwo,
                            (private  int)bb
             :      ClassTwo.bb            is    bb,
                    ClassThree.bb          is    bb,
```

$$\begin{array}{lll} \text{ClassThree.cc} & \text{is} & \ldots, \\ \text{ClassThree.methodBB} & \text{is} & \ldots, \\ \text{ClassThree.methodCC} & \text{is} & \ldots, \\ \text{ClassThree.methodCC} & \text{for} & \text{propertyName;} \end{array}$$

is

- Like:

```
class SomeClass{
    private int bb, bb;
    ⋮
}
```

we

- Allow
  - It,
    - ▹ Since:

      ClassThree *and* ClassTwo

will

- Be
  - Joined
    - ▹ Together
      - At:

        dd *and* state1
- In.

```
class SomeClass{
    ClassThree, ClassTwo      for   this.class;
    ClassThree.bb             is    dd;
    ClassTwo.cc               is    dd;
    ClassTwo.state1           is    ClassThree.state1;
    ClassThree.methodCC       is    newName;
    void voidReturner()       for   public;
    private int dd;
    // The bodies of: ClassThree and ClassTwo
    // will be copied here. And so if we write:
    // int i                  =     methodCC(8);
    // we will be refering to:
    // int methodCC(int); of: ClassTwo.
    // And similarly, for.
    // int i                  =     newName(8);
    // But their private members cannot be accessed here.
    // And if we write:
    // this.class.*           for   ClassTwo;
    // all things copied from those two classes
    // will be considered as a part of this class.
    // And we can write:
    // ClassTwo.public.state    is    protected;
    // and: ClassTwo(...) in the constructor.
    protected void voidReturner(){
        ClassTwo.state1          +=   {...};
    }
    ⋮
}
```

- But:

```
class   MixClassNine   =   ClassThree,   ClassTwo
                :    ClassThree.bb          is   bb,
                     ClassTwo.bb            is   bb,
                     ClassThree.cc          is   ...,
                     ClassThree.methodBB    is   ...,
                     ClassThree.methodCC    is   ...;
```

will

- Not

- ○ Compile.
  - ▷ And:

```
class SomeClass{
    public   ClassOne   o1,   o2   :   ((|int) newMethod)o1,
                                        (int d)o1,
                                        (int e)o2,
                                        (int a, b)o1;
    // But we cannot write: (int c)o1,
    // since: o1.c is visible in the environment.
    public SomeClassFour(){
        void   f7(int)          =   (i){...};
        o1.newMethod            =   f7;
    }
    public void voidReturner(){
        int   f8(int)           =   (i){ return o1.methodC(i); };
        int   f9(int)           =   (i){...};
        int   f10(int)          =   (i){...};
        o1.methodC              >=  f8;
        if (...) o1.methodC     =   f9;
```

```
                o1.methodC           >=  f10;

                o1.d                 =   o1.methodC(80);

            }

        }
```

will

- Be
  - Converted
    - To:

```
class  SomeClass{
    public     (int|int)     h      =     (i){  return  o1.methodC(i);  };
    private    int           o1Dota;
    private    int           o1Dotb;
    private    int           o1Dotd;
    private    (|int)        newMethod;
    private    int           o2Dote;
    public     ClassOne      o1,    o2;
    public  SomeClassFour(){
        int    f7(int)              =     (i){...};
        newMethod                   =     f7;
    }
    public  void  voidReturner(){
        void    f8(int)             =     (i){  return  h(i);  };
        int    f9(int)              =     (i){...};
        int    f10(int)             =     (i){...};
        h                           >=    f8;
        if    (...)    h            =     f9;
        h                           >=    f10;
```

```
            o1Dotd            =    h(80);
        }
    }
```

- And:

```
    SomeClass    sc    =    ...;
    ClassOne     co    =    sc.o1;
    int          i     =    sc.o1.methodC(1)  +  co.methodC(1);
```

- To:

```
    SomeClass    sc    =    ...;
    ClassOne     o     =    sc.o1;
    int          i     =    sc.h(1)  +  co.methodC(1);
```

- And:

$$(\text{int}\ \ a,\ \ b)o1$$

is

- Equivalent
  - To:

$$(\text{private}\ \ \text{int}\ \ a,\ \ b)o1.$$

- And

- If
    - ▹ We
        - Write:

(protected int e)o1,

- Then:

o1.e

will

- Be
    - Visible
        - ▹ In:
            - Subclasses.
- And
    - Similarly,
        - ▹ For:

(public int f)o1.

- And
    - Final
        - ▹ And partial:
            - Class-instances

cannot

- Be
    - Modified
        - ▹ This:
            - Way.
- Let.

```
class ClassFive{
    public  <T>            i,      j;
    public  [<U>  <T>]     k1,     k2;
    public ClassFive(){}
    public ClassFive(<T> a,  <T> b){
        i           =      a;
        j           =      b;
    }
    public <T> someMethod(<T> a){
        <T>   m     =      a   *   j;
        i           =      m   *   2;
        j           =      m   +   i;
        return i + j;
    }
    public static <T> staticMethod(<U> c){...}
    public static void voidReturner(<U> c){...}
}
```

- Then
    - We

- ▹ Can
  - – Write:

```
ClassFive   o1   =   new  ClassFive(100,  200)
                           :  ClassFive.T,
                              ClassFive.U   is  int;

ClassFive   o2   =   new  ClassFive()
                           :  ClassFive.T   is  int,
                              ClassFive.U   is  int;

ClassFive   o3   =   new  ClassFive()
                           :  ClassFive.T   is  int,
                              ClassFive.U   is  float;

ClassFive   o4   =   . . .  :   T,   U   is   int;

ClassFive   o5   =   . . .  :   T   is   int,   U   is   int;

ClassFive   o6   =   . . .  :   T   is   int,   U   is   float;

ClassFive   o7,  o8         :   T,   U   is   . . .;

int         i    =   ClassFive.staticMethod(1000)
                            :   T,   U   is   int;
```

- And
  - ○ If
    - ▹ In:
      - – An expression,

we

- Have
  - ○ To

▹ Use:

staticMethod

in

- Which
    - One
        - ▹ Instance
            - Uses:

long,

- And
    - The
        - ▹ Other:

int,

we

- Write
    - Two:
        - ▹ Statements,
- And
    - Then
        - ▹ Combine:
            - Them

in

- Another.
    - Exemplifying:

$$\text{long} \quad \text{i} \quad = \quad \text{ClassFive.staticMethod}(8) \quad : \quad \text{T, U is long;}$$

$$\text{int} \quad \text{j} \quad = \quad \text{ClassFive.staticMethod}(9) \quad : \quad \text{T, U is int;}$$

$$\text{long} \quad \text{k} \quad = \quad \text{i} + \text{j;}$$

- And
  - We
    - Can
      - Write:

$$\text{ClassFive} \quad \text{o1} \quad = \quad \text{null} \quad : \quad \text{T is int;}$$

$$\text{ClassFive} \quad \text{o2} \quad = \quad \text{new ClassFive()} \quad : \quad \text{T is int;}$$

$$\text{o1} \quad = \quad \text{o2;}$$

- But
  - Not:

$$\text{ClassFive} \quad \text{o1} \quad = \quad \ldots \quad : \quad \text{T is int;}$$

$$\text{ClassFive} \quad \text{o2} \quad = \quad \ldots \quad : \quad \text{T is float;}$$

$$\text{o1} \quad = \quad \text{o2;}$$

- Since:

$$\text{o1.T} \qquad \textit{and} \qquad \text{o2.T}$$

are

- Of

        - ◦ Different:
            - ▹ Types.

- And
    - ◦ We
        - ▹ Do not
            - Allow:

```
ClassFive  c5  =  ...  :  T  is  int;
c5             =  ...  :  T  is  float;
```

- Or
    - ◦ We
        - ▹ Do:
            - Not

allow

- Re-is-ing.
    - ◦ And
        - ▹ We:
            - Do

not

- Allow.

```
ClassFive  c5  =  new  ClassFive();
c5.i           =  ...               :  T  is  int;
```

- But
    - We
        - Can
            - Write:

```
ClassFive.T   is   int,   ClassFive.U   is   int;

ClassFive   c5   =   new  ClassFive();
```

- Or:

```
ClassFive.T,   ClassFive.U   is   int;

ClassFive   c5   =   new  ClassFive();
```

- Or:

```
ClassFive.U   is   int;

ClassFive.T   is   int;

ClassFive   c5   =   new  ClassFive();
```

- Or:

```
T,   U     is     int;

ClassFive   c5   =   new  ClassFive();
```

- And

- In:

  ClassFive.voidReturner(100) : T, U is int;

  ClassFive.T, ClassFive.U is int;

  ClassFive.voidReturner(100);

- When:

$$\text{ClassFive.voidReturner}(100) \quad : \quad \text{T,} \quad \text{U} \quad \text{is} \quad \text{int;} \qquad (93)$$

is

- Executed,
  - First:

T *and* U

- Of:

ClassFive

will

- Temporarily
  - Become:

int,

- And
  - Statement 93
    - ▷ Will:
      - Be executed.
- But

  - When:

ClassFive.T, ClassFive.U is int;

is

- Executed:

T *and* U

will

- Permanently
  - Become:

int,

- And:

ClassFive.voidReturner(100);

will

- Be
  - Executed,
    - ▷ And
      - In:

```
ClassFive.T,   ClassFive.U        is     int;

ClassFive.voidReturner(100);

ClassFive.U                       is     string;

ClassFive.voidReturner("xyz");
```

- When:

ClassFive.T, ClassFive.U is int;

is

- Executed:

T *and* U

will

- Become:

int,

- And
  - Later
    - ▹ When:

ClassFive.U is string;

is

- Executed:

U

will

- Become:

string,

- And:

T

will

- Remain
  - As:

▹ Such.

- But
  - In:

ClassFive.T, ClassFive.U is int;

ClassFive.voidReturner("xyz") : U is string;

ClassFive.voidReturner(50);

- First:

T *and* U

will

- Become:

int.

- And
  - When:

ClassFive.voidReturner("xyz") : U is string; (94)

is

- Executed:

U

will

- Temporarily
  - Become:

string,

- And
    - Statement 94
        - Will:
            - Be executed.
- And
    - After
        - That:

U

will

- Return
    - Back
        - To:

int.

- And
    - So:

ClassFive.voidReturner(10) : T, U is int;

ClassFive.voidReturner(10);

will

- Throw
    - An:
        - Exception.
- And

    - If:

```
ClassFive.T     is   int;
if (...){
   ClassFive.U  is   int;
   ClassFive.voidReturner(10);
}
ClassFive.voidReturner(20);
```

- Then:

ClassFive.voidReturner(10);

will

- Execute
    - Properly,
        - But:

ClassFive.voidReturner(20);

will

- Throw
    - An:
        - Exception.
- And
    - In:

ClassFive.T   is   int;

ClassFive.U   is   <T>;

ClassFive.voidReturner(10);

- When:

ClassFive.U   is   <T>;

is

- Executed:

U

will

- Be
  - Renamed
    - To:

T.

- And
  - Then
    - Since:

T

has

- Been
  - Is-ed
    - To:

int,

- We see that:

U

will

- Be
    - Is-ed
        - To:

int.

- Let.

```
class ClassSix{
    public ClassSix(){}
    public static <T> someMethod(){...}
    public static int staticMethod(<T> i){...}
}
class ClassSeven{
    public ClassSeven(){}
    public static <T> someMethod(){...}
    public static int staticMethod(<T> i){...}
}
```

- Then
  - We
    - Can
      - Write.

```
float        f   =   ClassSix.staticMethod(0)
                         +  ClassSeven.staticMethod(0.0f)
                            :   ClassSix.T        is    int,
                                ClassSeven.T      is    float;

int          i   =   ClassSix.staticMethod(0)
                         +  ClassSeven.staticMethod(0)
                            :   ClassSix.T,  ClassSeven.T
                                                  is    int;

i                =   ClassSix.staticMethod(0)
                         +  ClassSeven.staticMethod(0)
                            :   T                 is    int;

T                is  int;

i                =   ClassSix.staticMethod(0)
                         +  ClassSeven.staticMethod(0);

ClassSix     c6  =   new  ClassSix();

ClassSeven   c7  =   new  ClassSeven();

//  Note  that,  since  we  wrote:  T  is  int;

//  T  of  both:  c6  and  c7  will  be  is-ed  to:  int.
```

- Let.

```
class  ClassEight{

    ClassFive.T,    ClassFive.U    is    int;

    //  Or:  T,    U               is    int;

    public  ClassEight(){}

    ⋮

}
```

- Then:

$$\text{ClassFive} \quad \text{c5} \quad = \quad \text{new} \;\; \text{ClassFive}() \quad : \quad \text{T}, \quad \text{U} \quad \text{is} \quad \text{int}; \tag{95}$$

can

- Be
  - Rewritten
    - ▷ As:

$$\text{ClassFive} \quad \text{c5} \quad = \quad \text{new} \;\; \text{ClassFive}();$$

- Inside:

ClassEight.

- But
  - Statement 95
    - ▷ Written
      - Inside:

ClassEight

will

- Not
  - Produce
    - Any:
      - Error.
- Let:

```
class ClassNine{
    <T>    i,    j;
    <U>    m,    n;
    public ClassNine(){}
    public <T> someMethod(<T> p){...}
}
```

- And
  - We
    - Write:

```
class ClassTen{
    public ClassTen(){}
    public void someMethod(ClassNine c9){...}
}
```

- Then:

T *and* U

- Of:

ClassNine

will

- Be
  - Added
    - To:

ClassTen,

since

- They
  - Where
    - Not:
      - Is-ed.
- And
  - So
    - We:
      - Do

not

- Write:

ClassTen c10 = . . . : ClassNine.T, ClassNine.U is int;

- But:

ClassTen c10 = . . . : ClassTen.T, ClassTen.U is int;

or

ClassTen c10 = . . . : T, U is int;

- And
  - All:

ClassNine.T *and* ClassNine.U

- Inside:

ClassTen

will

- Be
  - Is-ed
    - ▷ To:

int.

- And
  - So
    - ▷ If:

```
class ClassEleven{
    ClassNine.T is int;
    public ClassEleven(){}
    public void someMethod(ClassNine c9){. . .}
}
```

we

- Write:

ClassEleven c11 = ... : U is int;

- And
  - Not:

ClassEleven c11 = ... : T, U is int;

- And
  - If:

```
class ClassTwelve{
    ClassNine.T is <U>;
    public ClassTwelve(){}
    public void someMethod(ClassNine c9){...}
}
```

- Then:

T

of

- All:

ClassNine

- Inside:

ClassTen,

will

- Be
  - Renamed
    - To:

U.

- And
  - So
    - We
      - Write:

ClassTwelve c12 = ... : U is int;

- And
  - Not:

ClassTwelve c12 = ... : T, U is int;

- And
  - Similarly,
    - If:

```
class ClassThirteen{
    <T> i, j;
    ClassNine.T   is   int;
    ClassNine.U   is   <V>;
    // Or: U      is   <V>;
    public ClassThirteen(){}
    public <T> someMethod(ClassNine c9){...}
}
class ClassFourteen{
    public ClassFourteen(){}
    public void someMethod(ClassThirteen c13){...}
}
```

we

- Write:

  ```
  ClassFourteen c14 = ... : T, V is int;
  ```

- And
  - Similarly,
    - For:

```
class ClassFifteen{
```

```
        ClassNine.T,    ClassNine.U    is    int;

        public  ClassFifteen(){}

        public  void  someMethod(ClassNine  c9){

            ClassNine.T                    is    <T>;

            ⋮

        }

    }
```

- And
  - If:

```
class  ClassSixteen{
    ClassNine.{T,  U}    is    int;
    public  ClassSixteen(){}
    public  ClassNine  someMethod(ClassNine  c9){
        ClassNine.U       is    string;
        ClassNine  cn     =     new  ClassNine();
        ⋮
    }
}
```

- Then:

T *and* U

in

- The
  - Parameter
    - ▷ And result
      - Of:

ClassNine someMethod(ClassNine);

will

- Be
  - Of
    - ▷ Type:

int,

- But:

cn.U

will

- Be
    - Of
        - ▹ Type:

string.

- And
    - So:

```
class  ClassSeventeen{
    public    <T>    i;
    public  ClassSeventeen(){}
    public  void  assign(ClassSeventeen  c17){  c17.i    =    8;  }
    public  <T>  methodTwo(){
        T                              is    int;
        ClassSeventeen    c17    =    ...;
        assign(c17);
        <T>                i        =    ...;
        return  i;
    }
}
```

should

- Be
    - Rewritten
        - As.

```
class  ClassSeventeen{
    public    <T>    i;
    public  ClassSeventeen(){}
    public  void  assign(ClassSeventeen  c17){  c17.i    =    8;  }
    public  <T>  methodTwo(){
        T                          is    int;
        ClassSeventeen    c17    =    . . . ;
        assign(c17);
        T                          is    <T>;
        <T>                  i       =    . . . ;
        return  i;
    }
}
```

- But:

ClassSeventeen.T

cannot

- Be
    - Renamed
        - Inside:

ClassSeventeen,

- Or
    - In
        - Its:
            - Subclasses.
- Let:

```
class ClassEighteen{
    public ClassEighteen(){}
    public <U> someMethod(<U> u){
        <T> t = new <T>();
        return t.methodOne(u);
    }
}
```

- And
    - Let:

```
class ClassNineteen extends ClassEighteen{
    public ClassNineteen(){}
    public <U> someMethod(<U> u){
        <T> t = new <T>();
        return t.methodTwo(u);
        }
    }
```

- And
  - Let.

```
    class ClassTwenty{
        public ClassTwenty(){}
        public void someMethod(){
            ClassEighteen   obj   =   new  ClassEighteen();
            obj                   =   new  ClassNineteen();
        }
    }
```

- Then
  - We see that,
    - ▷ The:
      - Compiler

can

- Understand
  - That:

T

- In:

```
ClassEighteen obj = new ClassEighteen();
```

- Requires:

```
<U> methodOne(<U>);
```

- And:

```
T
```

- In:

```
obj = new ClassNineteen();
```

- Requires:

```
<U> methodTwo(<U>);
```

- And
  - So:

```
ClassTwenty.T
```

- Requires:

```
<U>    methodOne(<U>);

<U>    methodTwo(<U>);
```

- And
  - So:

```
ClassTwenty c20 = new ClassTwenty()
                       : U is int,
                         T is SomeClass;
```

will

- Not
  - Compile,
    - ▷ If:

SomeClass

does

- Not
  - Have:

int methodOne(int); *and* int methodTwo(int);

- But
  - If:

```
class SomeClass{
    public SomeClass(){}
    public int methodOne(int i){...}
    public int methodTwo(int i){...}
    public float methodTwo(int i){...}
}
```

- Then:

U

- In:

ClassTwenty c20 = new ClassTwenty() : T is SomeClass;

will

- Be
  - Is-ed
    - ▷ To:

int.

- And
  - If:

```
class SomeClass{
    public SomeClass(){}
    public int methodOne(int i){...}
    public int methodTwo(int i){...}
    public float methodOne(int i){...}
    public float methodTwo(int i){...}
}
```

- Then:

U

will

- Be
  - Left

        - ▷ Dangling.

- And
    - Required
        - ▷ In:
            - The class

used

- For
    - Is-ing
        - ▷ Will:
            - Also

be

- Generated
    - By:
        - ▷ Documentation
            - Tools.

- And:

T is SomeClass;

can

- Be
    - Written
        - ▷ Only:
            - If

the

- Default
    - Constructor

- ▷ Of:

SomeClass

is

- Visible
  - In
    - ▷ The:
      - Environment.
- And
  - If:

```
interface InterfaceOne{
    public <T> someMethod(<T> a);
}
```

we

- Can
  - Write:

```
class ClassTwentyOne implement InterfaceOne{
    public ClassTwentyOne(){}
    public <T> someMethod(<T> a){
        // This is the implementation
        // of the method in: InterfaceOne.
        ⋮
    }
}
```

- Or:

```
class ClassTwentyOne implement InterfaceOne{
    InterfaceOne.T    is    int;
    // Or: T          is    int;
    public ClassTwentyOne(){}
    public int someMethod(int a){
        // This will be the implementation
        // of the method in: InterfaceOne,
        // since we wrote: InterfaceOne.T is int;
        ⋮
    }
}
```

- And
  - Statements
    - ▷ Like:

```
ClassNine.T,   ClassNine.U        is        int;
ClassNine.T                       is        <V>;
```

cannot

- Be:

static *or* public *or* protected *or* private.

- And
    - If:

      (<T>|) ft = (){ return new <T>(. . .); };

- And
    - We
        - ▹ Is:

          T *to*: int,

- Then:

  “*default-value*”

will

- Be
    - Used
        - ▹ Instead
            - Of:

              new int(. . .).

- And
    - We
        - ▹ Can
            - Write:

```
(enum)ClassFive   cfz   =   . . .   :   T   is   int;

[ClassFive]       cfl   =   . . .   :   T   is   int;

ClassFive [ ]     cfa   =   . . .   :   T   is   int;
```

- And:

```
@("...")
package subPackage extends superPackage   :  T  is  int;

@("...")
import somePackage.∗                       :  T  is  int;

@("...")
import SomeClass                           :  T  is  int;

⋮
```

is

- Equivalent
  - To.

```
@("...")
package subPackage extends superPackage;

T is int;

@("...")
import somePackage.∗;

@("...")
import SomeClass;

// If we wrote: superPackage.T is int;

// then only: superPackage.T == int.

⋮
```

- Let:

sc sc1 *and* sc2

be

- Instances
  - Of:

```
public class SomeClass{
    public [int] k5;
    k5 for inbox;
    public SomeClass(){}
    public int intReturner(int i){...}
    public boolean boolReturner(){...}
    ⋮
}
```

- And
  - Let:

scl

be

- An
  - Instance
    - ▹ Of:

[SomeClass],

- And
  - Let:

scli1 *and* scli2

be

- Instances
  - Of:

[SomeClass int],

- Then
  - We
    - ▹ Can
      - Write:

sc1 = new sc2;

for

- Cloning.
  - And:

int int t = new i j;

is

- Equivalent
  - To:

int int t = i j;

- And

    ◦ If:

$$\text{scli1} \quad = \quad \text{new} \quad \text{scli2;}$$

- Then:

“*for all*: i, scli1[i] == new scli2[i][0].”

- And
    ◦ Similarly,
        ▹ For:

trees and arrays.

- The
    ◦ Interpretation
        ▹ Of:

$$\text{this} \quad += \quad \text{sc;} \tag{96}$$

- Is:

“*add*: sc *to the program pool.*”

- Note that,
    ◦ Since:

sc

is

- Added
    ◦ To
        ▹ The
            – Program-pool:

this

in

- Statement 96
    - Means,
        - ▹ The:
            - Program,
- And
    - Not
        - ▹ The:
            - Object

which

- Executed
    - That:
        - ▹ Statement.
- And
    - Statement 96
        - ▹ Will:
            - Throw

an

- Execption,
    - If:

sc

has

- Not
    - Been
        - ▹ Fully:

        - Initialized.

- The
    - Interpretation
        - Of:

$$\text{int} \quad i \quad = \quad \text{this[SomeClass][0].intReturner(8);} \qquad (97)$$

- Is:

"*if there is an idle instance of*: SomeClass *in the program pool,*

then invoke: int intReturner(int); *of that object.*"

- And
    - Statement 97
        - Will:
            - Throw

an

- Exception,
    - If
        - There:
            - Is

no

- Idle
    - Instance.
        - The
            - Interpretation of:

$$\text{boolean} \quad b \quad = \quad \text{this[SomeClass][0];}$$

- Is:

"*is there at least one idle instance of*: SomeClass?"

- And
    - That
        - ▷ Of:

```
boolean b = this[(extends)SomeClass][0];
```

- Is:

"*is there at least one idle instance of a subclass of*: SomeClass?"

- And
    - That
        - ▷ Of:

```
boolean b = this[(class)SomeClass][0];
```

- Is:

"*is there at least one idle instance of*: SomeClass *or its subclass*?"

- And
    - That
        - ▷ Of:

```
boolean b = this[SomeClass][ ];
```

- Is:

"*are all instances of*: SomeClass *idle*?"

- And
    - That
        - ▷ Of:

$$\text{int} \quad \text{i} \quad = \quad (\text{this[SomeClass][ ]}).\text{length};$$

- Is:

"*get the number of all idle instances of*: SomeClass."

- And
  - Similarly,
    - ▹ For.

$$\text{scl} \quad = \quad \text{this[SomeClass][ ]} \quad : \quad (\text{this[SomeClass][ ]}.\text{boolReturner}());$$

- The
  - Interpretation
    - ▹ Of:

$$\text{int} \quad \text{i} \quad = \quad (\text{this[ ][ ]}).\text{length};$$

- Is:

"*get the number of idle objects*,"

- And
  - That
    - ▹ Of:

$$\text{this} \quad \%= \quad \text{SomeClass};$$

- Is:

"*remove an idle object of type*: SomeClass."

- And
  - Similarly,
    - ▹ For.

```
this        =     sc1;
int    i    =     this[(class)SomeClass][0].intReturner(8);
this        %=    (class)SomeClass;
this        =     ;
```

- The
  - Interpretation
    - Of:

```
boolean   b   =   this[SomeClass][+];
```

- Is:

  "*is there at least one busy instance of*:   SomeClass?"

- And
  - That
    - Of:

```
int   i   =   (this[SomeClass][+]).length;
```

- Is:

  "*get the number of busy instances of*:   SomeClass,"

- And
  - Only:
    - Volatile
      - Methods

can

- Be

- Used
  - ▷ In.

$$\text{i} = (\text{this[SomeClass][+]} : \text{this[SomeClass][+].<method>()}).\text{length};$$

- The
  - Interpretation
    - ▷ Of:

$$\text{this[SomeClass][+].inbox} = 8; \tag{98}$$

- Is:

"*put*: 8 *into*: inbox *of all busy instances of*: SomeClass."

- And
  - Similarly,
    - ▷ For.

$$\text{this[SomeClass][+].inbox} = 8 : (\text{this[SomeClass][+].<method>()});$$

- The
  - Interpretation
    - ▷ Of:

$$\text{this[SomeClass][}*\text{].inbox} = 8;$$

- Is:

"*put*: 8 *into*: inbox *of all instances of*: SomeClass."

- And
  - Similarly,
    - ▷ For:

$$\text{this[SomeClass][}*\text{].inbox} = 8 : (\text{this[SomeClass][}*\text{].<method>()});$$

- And
    - These
        - Statements:
            - Can

also

- Be
    - Executed
        - In:
            - Objects

that

- Resisdes
    - In
        - The:
            - Pool.
- And
    - If
        - An:
            - Object

in

- The pool
    - Executes
        - Statement 98,
            - Then:

8

will

- Not
  - Be
    - Put
      - Into:

inbox

of

- That
  - Object.
    - And:
      - Serialization

will

- Be
  - Done
    - Automatically:
      - If required.
- And
  - If:

$$(\text{int}|\text{int}) \quad f1 = (i)\{\ldots\}, \quad f2 = (i)\{\ldots\};$$

$$\text{sc.inbox} = f1;$$

$$\text{sc.inbox} += f2;$$

- And
  - An
    - Element:

– Has

been

- Put
  - Into:

sc.inbox,

- And
  - If:

sc

is

- Idle,
  - Then
    - ▷ That:
      - – Element

will

- Be
  - Removed:
    - ▷ From

the

- Associated
  - List,
    - ▷ And:
      - – Will

be

- Given

- To:

$$\text{f1} \quad \textit{and} \quad \text{f2}$$

after

- Cloning,
  - And
    - If:

$$\text{sc}$$

is

- Busy,
  - This
    - Will:
      - Be done

when

- It
  - Becomes:
    - Idle.
- And
  - We
    - Can
      - Write:

$$\text{sc.inbox} = \text{sc.inbox}[\ ..\ 8],\quad \text{sc.inbox}[9\ ..\ ];$$

- And
  - We
    - Do not

– Allow:

$$\text{this} \quad += \quad \text{this;}$$

- And:

$$\text{sc} \;\; \text{instanceof} \;\; \text{SomeClass}$$

can

- Be
  - Rewritten
    - ▷ As:

$$\text{sc.class} \quad == \quad \text{SomeClass}. \tag{99}$$

- And:

$$(\text{sc.class} \quad > \quad \text{SomeClass}) \quad == \quad \text{true}, \tag{100}$$

- If:

$$\text{sc}$$

is

- An
  - Instance:
    - ▷ Of

a

- Subclass
  - Of:

$$\text{SomeClass}.$$

- And:

$$\text{sc1.class} \quad == \quad \text{sc2.class} \qquad \textit{and} \qquad \text{sc1.class} \quad > \quad \text{sc2.class}$$

are

- Like
  - Expressions 99 and 100.
    - ▹ And
      - If:

$$\text{ClassFive} \quad \text{c5} \quad = \quad \ldots \quad : \quad \text{T} \quad \text{is} \quad \text{int;}$$

- Then:

$$\text{c5.T} \quad == \quad \text{int.} \tag{101}$$

- And
  - We
    - ▹ Do not
      - Write:

$$\text{c5.T.class} \quad == \quad \text{int}$$

since

- Expression 101
  - Is
    - ▹ Like:

$$\text{int} \quad == \quad \text{int.}$$

- The
  - Interpretation
    - ▹ Of:

$$\text{string} \quad \text{s} \quad = \quad \text{sc.class;}$$

- Is:

"*get the name of the class of*: sc,"

- And
  - That
    - ▷ Of:

string s = sc.super[0];

- Is:

"*get the name of the immediate superclass of*: sc,"

- And
  - That
    - ▷ Of:

[string] sl = sc.super[ ];

- Is:

[string] sl = sc.super[0], sc.super[1], . . . ;

- And
  - That
    - ▷ Of:

[string] sl = sc.class.interface;

- Is:

"*get the names of all interfaces implemented by the class of*: sc,"

- And
  - That
    - ▷ Of:

[string] sl = sc.interface;

- Is:

[string] sl = sc.class.interface, sc.super[0].interface, . . . ;

- And
    - That
        - ▹ Of:

[import string] isl = sc.class.fields[public];

- Is:

"*get the details of all public fields in the class of*: sc."

- And
    - Similarly,
        - ▹ Using:

sc.class.fields[public.static] *and* sc.class.fields[protected].

- The
    - Interpretation
        - ▹ Of:

[import string] isl = sc.class.fields[ ];

- Is:

"*get the details of all fields in the class of*: sc,"

- And
    - Similarly,
        - ▹ For:

[import string] isl = sc.class.methods[ ], sc.super[0].methods[ ];

- And
  - If:

```
[int]   k5   =   scl[ ].class  ==  SuperClass;

[int]   k6   =   scl[ ].class   >  SuperClass;
```

- Then:

k5

will

- Hold
  - The
    - Indices:
      - Of

all

- Instances
  - Of:

SuperClass,

- And:

k6

the

- Indices
  - Of

▷ All:

– Instances

of

- Subclasses
  - Of:

SuperClass.

- The
  - Interpretation
    ▷ Of:

boolean b = scl[ ].class == SuperClass;

- Is:

"*are all objects in*: scl *instances of*: SuperClass?"

- And
  - That
    ▷ Of:

b = SuperClass in scl[ ].class;

- Is:

"*does*: scl *contain an instance of*: SuperClass?"

- Let:

thread *and* ithread

be

- Keywords.
  - And

            ▷ Let:

```
ithread SomeIThread{
    ⋮
}
public thread ThreadOne implements SomeIThread{
    public ThreadOne(){...}
    public int reader(int i){...}
}
```

- And
    - ◦ Let.

```
public thread ThreadTwo{
    public ThreadTwo(){...}
    public int writer(){...}
}
```

- Then
    - ◦ We
        - ▷ Can
            - Write:

```
ThreadOne   to   =   new ThreadOne();

if (to == null){...}
```

- But
  - Not:

```
ThreadOne   to   =   ...;

int         i    =   to.reader(10);
```

- Or
  - We
    - ▷ Can:
      - Create

any

- Number
  - Of:
    - ▷ Thread
      - Instances,
- But
  - We
    - ▷ Cannot:
      - Directly access

any

- Non
  - Static:

        ▷ Thread
            – Member.

- Let:

```
public class SomeClass{
    public    int          i;
    public    ThreadOne    to;
    public    ThreadTwo    tt;
    to.process    =    {
                             i    −=    read(i);
                             // We do not write: to.read.
                         };
    tt.process    =    {
                             i    +=    write();
                         };
    ⋮
}
```

- And
  - We
    - ▷ Execute:

$$\text{do process;} \tag{102}$$

then

- All
  - Thread

▹ Associated
  – With:

process,

in

- This
  - Case:

to *and* tt,

will

- Be
  - Put
    ▹ Into:
      – A queue,
- And
  - After
    ▹ That:
      – If

the

- First
  - Thread
    ▹ Can:
      – Lock

all

- Things it
  - Should:
    ▹ Read

      - Or write,

in

- Its
    - Process-block,
        - ▹ Then:
            - It

will

- Lock
    - All
        - ▹ Of:
            - Them,
- And
    - Execute
        - ▹ All:
            - Statements

in

- Its:
    - Process
        - ▹ Block,
- And
    - Then
        - ▹ Release:
            - All locks,
- And
    - Goto

        - ▹ The:
            - End

of

- The
    - ◦ Queue,
        - ▹ And:
            - So forth,

until

- We
    - ◦ Execute.

$$\text{do} \quad \text{!process;} \tag{103}$$

- Let:

```
public    ThreadOne    to;
public    ThreadTwo    tt;
to.pr1.subPr1.init     =     {
                                 ⋮
                                 do  subPr2;
                             };
to.pr1.subPr2          =     {
                                 ⋮
                                 return;
                             };
tt.pr1.subPr3.init     =     {
                                 ⋮
                                 do  subPr4;
                             };
tt.pr1.subPr4          =     {};
tt.pr1.subPr5          =     {...};
```

- And
  - We
    - ▷ Execute:

do pr1;

- Then:

to *and* tt

will

- Seek
  - To
    - Perform:

subPr1 *and* subPr3

respectively.

- And
  - After:

to

- Finishes:

subPr1,

it

- Will
  - Seek
    - To
      - Perform:

subPr2,

- And
  - After
    - That:

            - It

will

- Exit
    - Out
        - Of:

pr1,

since

- We
    - Wrote:

return;

- In:

subPr2.

- And
    - After:

tt

- Finishes:

subPr3,

it

- Will
    - Seek
        - To
            - Perform:

subPr4,

- And
    - After
        - ▷ That:
            - It

will

- Not
    - Perform:

subPr5,

- But
    - Start from
        - ▷ Its:
            - Initial point,

since

- We
    - Did
        - ▷ Not
            - Write:

do subPr5; *or* return;

- In:

subPr4.

- Let:

```
public    ThreadOne    t1,    t2,    t3;

t1.pr1        =        {...};

t2.pr1        =        {...};

t2.pr2        =        {...};

t3.pr2        =        {...};

t3.pr3        =        {...};
```

- And
    - We
        - Execute:

do pr1;

- Then:

t1 *and* t2

will

- Perform:

pr1

until

- We
    - Execute:

do !pr1;

- And
    - After:

$$\text{pr1}$$

is

- Over,
  - ◦ We
    - ▹ Can
      - – Execute.

$$\text{do} \quad \text{pr2}; \tag{104}$$

- But
  - ◦ If we
    - ▹ Execute statement 104
      - – While:

$$\text{pr1}$$

is

- Going
  - ◦ On,
    - ▹ Then:
      - – There

will

- Be
  - ◦ An
    - ▹ Exception,
      - – Since:

$$\text{pr1} \qquad \textit{and} \qquad \text{pr2}$$

- Shares:

t2.

- And
  - We
    - Can
      - Execute:

```
boolean b = pr1;
```

to

- Check
  - Whether:

pr1

is

- Going
  - On
    - Or:
      - Not.
- The
  - Interpretation
    - Of:

```
do !pr1, pr2;
```

- Is:

"*stop*: pr1, *and start*: pr2,"

- And
  - That

        - ▹ Of:

```
do   pr1,   pr3;
```

- Is:

  “*simultaneously start*:   pr1   *and*   pr3.”

- And
    - If:

```
public  class  SuperClass{
    public      int           i;
    public      ThreadOne     t1,   t3;
    private     ThreadOne     t2;
    t1.pr1.subPr1                   =   {...};
    t1.pr1.subPr2                   =   {...};
    t2.pr1                          =   {...};
    t2.pr2                          =   {...};
    t3.pr3                          =   {...};
    public      transient     seq1  =   voidReturner(i)  >  pr2;
    public      transient     seq2  =   (seq1  :  ?  bool-Exp);
    public      transient     seq3  =   pr1  >  seq2  >  pr3;
    void  voidReturner(int  i){...}
    ⋮
}
```

- And
  - If:

```
public class SubClass extends SuperClass{
    t1.pr1.subPr2 = {...};
    public SubClass(){}
    public void voidReturner(){ do seq3; }
}
```

we

- Can
  - Write:

```
[SubClass] scl = ...;
do scl[ ].seq1, scl[ ].pr3;
```

- But
  - Not:

```
[SubClass] scl = ...;
do scl[ ].pr2;
```

- Or
  - If
    - ▷ All:
      - Participating threads

of

- A process
  - Is:

private,

then

- That
  - Process
    - ▹ Will
      - Be:

private,

- And
  - If
    - ▹ One:
      - Of

the

- Threads
  - Is:

protected,

then

- That
  - Process
    - ▹ Will:
      - Be

at

- Least:

protected,

- And
  - Similarly,
    - ▷ For:

public.

- Let:

```
public ThreadOne t1, t2;
t1.process = {
                 ⋮
                 break thread;
             };
t2.process = {
                 ⋮
                 continue t1;
                 // There will be a compilation error,
                 // if: t1 does not belong to
                 // this process.
                 // And if there is ambiguity,
                 // we write: continue this.t1;
                 ⋮
             };
```

- And
  - When:

```
break thread;
```

is

- Executed:

```
t1
```

will

- Wait
    - To
        - Be:
            - Notified,
- And
    - When:

```
continue t1;
```

is

- Executed,
    - It
        - Will:
            - Be notified.
- And
    - If
        - We
            - Execute:

```
continue do;
```

all

- Threads
    - Of
        - The:

- Process

that

- Are
  - Waiting
    - For:
      - Notification

will

- Be
  - Wokenup.
    - And
      - If:

$$\text{break thread}(8); \qquad \textit{or} \qquad \text{break thread}(8,\ 9);$$

is

- Executed,
  - The
    - Corresponding:
      - Thread

will

- Wait
  - For:

$$8 * 100000 \quad \textit{nanoseconds} \quad \textit{or} \quad 8 * 100000 + 9 \quad \textit{nanoseconds}$$
$$\textit{respectively.}$$

- Assume
  - That:

$$\text{do} \quad \text{childProcess;} \qquad (105)$$

was

- Executed
    - In.

$$\text{t1.parentProcess} \quad = \quad \{\ldots\};$$

- Then
    - If
        - ▹ We
            - Execute:

$$\text{continue} \quad \text{thread.outer;}$$

- In:

$$\text{childProcess,}$$

the

- Thread
    - Which
        - ▹ Executed:
            - Statement 105,
- Or:

$$\text{t1}$$

will

- Be
    - Notified.
        - ▹ And:
            - If

we

- Execute:

$$\text{continue do.outer;} \tag{106}$$

all

- Threads
    - Of
        - The:
            - Parent-process

will

- Be
    - Wokenup.
        - And:
            - Statement 106

will

- Be ignored
    - In
        - Processes:
            - Started

by

- The
    - Host
        - Class.
            - Let:

```
public class SomeClass{
    public static int staticField;
    ⋮
}
```

- And
  - Let:

```
public thread ThreadThree{
    public ThreadThree(){}
    public void voidReturner(){ SomeClass.staticField++; }
}
```

- And
  - Let.

```
ThreadThree tt;

tt.process = { voidReturner(); };
```

- Then
  - Variables
    - Used
      - In:

```
void voidReturner();
```

- Or:

SomeClass.staticField (107)

will

- Not
  - Be:
    - Locked.
- And
  - So
    - The:
      - Value

in

- Variable 107
  - May
    - Not:
      - Be consistent.
- Or

- Only
  - ▷ Fields:
    - That

are

- Explicitly
  - Accessed
    - ▷ Inside:
      - That block

will

- Be
  - Locked.
    - ▷ And so
      - If:

```
ThreadThree tt;

tt.process = { if (boolReturner(...)){...} };

boolean boolReturner(int i){...}
```

then

- We
  - Should
    - ▷ Take:
      - Care

not

- To

- Access
  - The:
    - Fields

of

- The
  - Class
    - In.

boolean boolReturner(int i){...}

- And
  - So
    - We:
      - Say

that,

- Only
  - Volatile
    - Methods:
      - Of

the

- Class
  - Can
    - Be:
      - Used

in

- Process
  - Blocks.
    - Let.

```
public thread ThreadFour{
    public SomeClass sc = null;
    public ThreadFour(){}
    public Threadfour(SomeClass sc){...}
    public volatile boolean volatileBoolReturner(){...}
    ⋮
}
```

- Then
  - We
    - Can
      - Write:

```
public    ThreadFour    tf1,    tf2

tf1.pr1    =    {
                  ⋮
                  if  (tf2.volatileBoolReturner()){. . . }
                  // Non volatile methods of other threads
                  // (whether of this or other processes)
                  // cannot accessed here.
                  // And non static fields of other thread
                  // are read only here.
             };

tf2.pr2    =    {. . . };
```

- And
  - Non
    - Static:
      - Thread-fields

will

- Be
  - Read
    - Only:
      - In

the

- Host
    - Class.
        - ▹ And
            - If:

```
ThreadFour     tf     :     ((int   i)tf;
```

- Then:

```
tf.i
```

will

- Be
    - Read
        - ▹ Only:
            - In

the

- Host
    - Class.
        - ▹ And:
            - So

we

- Can
    - Invoke:
        - ▹ Volatile
            - Thread-methods

in

- The:
    - Host

            ▷ Class.

- And
    - If
        ▷ We
            - Execute:

```
SomeClass    sc    =    . . . ;

ThreadFour   tf    =    new  Threadfour(sc);
```

a

- Clone
    - Of:

sc

will

- Be
    - Used
        ▷ In:

```
ThreadFour    tf    =    new  ThreadFour(sc);
```

- Or
    - It
        ▷ Is:
            - Like

saying

- That,

- Call
    - By:
        - Reference

is

- Not
    - Allowed
        - With:
            - Threads,

so that,

- We
    - Can
        - Avoid:
            - Deadlocks.
- But
    - In:

```
ThreadFour    tf;

tf.process    =    {...};
```

all

- Objects
    - Given
        - To:

tf

will

- Not
  - Be:
    - ▷ Cloned.
- And
  - If
    - ▷ We
      - Write:

$$\text{someList}[\,][0]{+}{+} \quad : \quad (\ldots);$$

- Then:

$$\text{someList},$$

- And
  - Not
    - ▷ Just:
      - Some locations

in

- That
  - List;
    - ▷ Will:
      - Be locked.
- And
  - If
    - ▷ We
      - Write:

$$\text{native}\{\dots\}; \tag{108}$$

then

- Native
  - Fields
    - ▷ Used:
      - In statement 108

will

- Not
  - Be
    - ▷ Locked:
      - Unless

they

- Are
  - Used
    - ▷ Outside:
      - Statement 108.
- Let:

$$\text{tf}, \qquad \text{tf1} \qquad \textit{and} \qquad \text{tf2}$$

be

- Instances
  - Of:

$$\text{ThreadFour}.$$

- Then:

$$\text{tf?} \;==\; 7,$$

- If:

$$\text{tf}$$

is

- Working,
  - And:

$$\text{tf? == 9,}$$

if

- It
  - Is
    - Waiting,
      - And:

$$\text{tf? == 10,}$$

if

- It
  - Is:
    - Inactive.
- And
  - We
    - Can
      - Write:

$$\text{int i = tf1;} \quad \textit{and} \quad \text{tf1 = 8;}$$

to

- Get

        - And
            - Set:
                - The priority.

- And:

$$\text{int} \quad i = \text{this}; \tag{109}$$

$$\text{this} = 8; \tag{110}$$

inside

- Threads
    - For
        - The:
            - Same.

- And
    - If
        - We
            - Execute:

$$\text{tf1} = \text{tf2};$$

the

- Priority
    - Of:

$$\text{tf2}$$

will

- Be
    - Given
        - To:

tf1.

- Or
    - We
        - ▹ Do:
            - Not

allow

- The
    - Address
        - ▹ Of:
            - A thread

to

- Be
    - Copied
        - ▹ Into:
            - Another.
- And
    - So
        - ▹ We:
            - Do

not

- Allow
    - Methods
        - ▹ Like:

```
<thread-name>  threadReturner(<thread-name>);
```

- And:

trees, *lists* *and* *arrays*

of

- Threads.
    - And
        - If:

```
void voidReturner(int);
```

is

- Some
    - Method,
        - Then:

```
voidReturner(tf);
```

is

- Equivalent
    - To:

```
int i = tf;
voidReturner(i);
```

- And
    - If
        - We
            - Write:

$$\text{tf.continue} = \{\ldots\};$$

then

- That
  - Block
    - ▷ Will:
      - – Be executed

when

- That
  - Thread
    - ▷ Is:
      - – Wokenup.
- And
  - If
    - ▷ We
      - – Write:

$$\text{tf.break} = (\text{i})\{ \quad \text{thread} = \text{i}; \quad \}; \tag{111}$$

- And
  - Another
    - ▷ Thread
      - – Preempts:

tf,

then

- That
  - Block
    - ▷ Will:
      - – Be executed

with

- The
  - Priority:
    - Of

the

- New
  - Thread
    - As:
      - Parameter.
- And
  - We
    - Use:

thread *instead of*: this

in

- Statement 111,
  - Since
    - It:
      - Is

written

- In
  - Classes.
    - And
      - If:

$$\text{tf.break} = \{\ldots\}; \tag{112}$$

then

- That

- Block
    - Will:
        - Be executed

when

- The
  - Thread:
    - Exits

its

- Critical
  - Section.
    - And:
      - Statement 112

will

- Also
  - Be
    - Executed
      - After:

```
break thread;
```

- And
  - We
    - Can:
      - Give

a

- Description
  - For:

$$\begin{aligned}
&\text{tf.<process-Name>.break} && = && (i)\{\ldots\}; \\
&\text{tf.<process-Name>.<subprocess-Name>.break} && = && (i)\{\ldots\}; \\
&\text{tf.<process-Name>.break} && = && \{\ldots\}; \\
&\text{tf.<process-Name>.<subprocess-Name>.break} && += && \{\ldots\};
\end{aligned}$$

like

- That
    - Which
        - We:
            - Did

in

- Section 2.
    - And
        - We
            - Can write.

```
ThreadOne   to   =   null;
to               =   new  ThreadOne();
```

- Let:

exception *and* iexception

be

- Keywords.
    - And

▹ Let:

```
iexception SomeIException{
    ⋮
}
exception SomeException implements SomeIException{
    ⋮
}
```

- And
  - ○ We
    - ▹ Execute:

```
SomeException   se1     =     null;
se1                     =     new SomeException();
se1                     =     "Error message.";
string          s       =     se1;
SomeException   se2     =     new SomeException();
SomeException   se3     =     se1;
se1                     =     (string)se2;
```

the

- Error
  - Message
    - ▹ In:

se2

will

- Be:

"",

- And
  - It
    - ▹ Will:
      - Be

given

- To:

se1.

- And
  - We
    - ▹ Can
      - Write:

$$\text{string} \quad s \quad = \quad \text{this;} \qquad (113)$$
$$\text{this} \quad = \quad \texttt{"Error message."}; \qquad (114)$$

inside

- Exceptions.
  - And
    - ▹ Statement 109:

          – Written

in

- Classes
    - And
        - ▹ Exceptions:
            - – Will

be

- Converted
    - To:

int i = 0;

- And
    - Statement 113:
        - ▹ Written

in

- Classes
    - And
        - ▹ Threads:
            - – Will

be

- Converted
    - To:

string s = "";

- And
    - Statement 110:

        - ▹ Written

in

- Classes
    - And:
        - ▹ Exceptions,
- And
    - Statement 114:
        - ▹ Written

in

- Classes
    - And:
        - ▹ Threads,
- And
    - Statements
        - ▹ Like:
            - Statements 96, 102 and 103
- And
    - Other
        - ▹ Similar:
            - Statements

written

- In
    - Threads
        - ▹ And:
            - Exceptions

will

- Be
  - Ignored.
    - ▹ And:
      - – Classes

cannot

- Extend:

exceptions *and* *threads*,

- Or
  - Implement:

iexceptions *and* *ithreads*,

- And
  - Similarly
    - ▹ For:
      - – Others.
- And
  - By
    - ▹ Default:

exceptions *and* *threads*

will

- Extend:

DefaultSuperException *and* DefaultSuperThread,

- And:

DefaultSuperException *and* DefaultSuperThread

will

- Not
  - Extend
    - Themselves.
      - Let:

```
boolean   b1(int);

boolean   b2(int);

boolean   b3(int);
```

be

- Methods,
  - And
    - Let.

```
abstract  a1  =  {

                     int  i  =  10;

                     c1(i)   :-   c2(i)  &&  b1(i);

                     c2(i)   :-   b2(i)  ||  c3(i);

                     c3(i)   :-   b3(i)  ||  c1(i);

                 };

abstract  a2  =  {
```

$$\mathrm{c1(i)} \quad :- \quad \ldots;$$

$$\vdots$$

$$\};$$

$$\text{abstract} \quad \text{a3} \quad = \quad \{\ldots\};$$

- Then
  - We
    - ▹ Can
      - Write:

$$\text{boolean} \quad \text{b} \quad = \quad \text{a1.c1}(10);$$

- And we
  - Say
    - ▹ That:
      - Clauses

that

- Does
  - Not
    - ▹ Contain:
      - Logical-operators

are

- Data
  - Clauses.
    - ▹ And
      - So:

$$\texttt{c1("abc");} \qquad \textit{and} \qquad \texttt{c2("efg");}$$

are

- Data
  - Clauses,
    - ▷ And:

$$\text{c1(i)} \quad :- \quad \ldots; \qquad \textit{and} \qquad \text{c2(i)} \quad :- \quad \ldots;$$

are

- Non
  - Data-clauses.
    - ▷ And:
      - We

say

- That,
  - After:
    - ▷ Initialization,

only

- Data
  - Clauses:
    - ▷ Can

be

- Added
  - Or
    - ▷ Remved:
      - From

these

- Variables.
  - And
    - So:
      - We

cannot

- Write:

$$a1 = \{ \; c5(i) \; :- \; \ldots; \; \};$$

- But
  - We
    - Can
      - Write:

$$a1 = \{ \; c1(\texttt{"abc"}); \; \};$$

- And
  - If:

$$a1 = a2;$$

then

- Non
  - Data
    - Clauses
      - In:

$$a1$$

will

- Remain

  - As:
    - ▹ Such,
- But
  - Data
    - ▹ Clauses
      - In:

a2

will

- Replace
  - Those
    - ▹ That:
      - Are

there

- In:

a1.

- And
  - If:

a1 += a2;

then

- All
  - Data
    - ▹ Clauses
      - In:

a2

will

- Be
  - Appended
    - To:

$$\text{a1}.$$

- And
  - We
    - Can
      - Write:

$$\text{a1} \quad = \quad \text{a2} \;\; || \;\; \text{a3};$$

- And
  - Similarly,
    - Using:

&&, %, &=, |= *and* %=.

- The
  - Interpretation
    - Of:

$$\text{int} \quad \text{i} \quad = \quad \text{a1};$$

- Is:

"*get the number of data-clauses in*: a1,"

- And
  - That
    - Of:

$$(0)\mathrm{a1};$$

- Is:

  "*remove repetitons and data-clauses that cannot be used in*: a1,"

- And
  - That
    - ▷ Of:

```
int   i   =   (0)a1;
```

- Is:

```
abstract   tempA   =   (0)a1;
int        i       =   tempA;
```

- And
  - Similarly,
    - ▷ For.

```
boolean   b   =   !(0)a1  ||  c80("abc")  !in  a1;
```

- Let:

```
class  SomeClass{
    protected   float   b1,  b2,  b3;
    protected   float   c1,  c2,  c3;
    protected   float   d1,  d2,  d3;
    protected   float   e1,  e2,  e3;
    public      float   x,   y,   z;
    public  abstract  a4  =  {
                                e1 = b1 ∗ x  +  c1 ∗ y  +  d1 ∗ z;
                                e2 = b2 ∗ x  +  c2 ∗ y  +  d2 ∗ z;
                                e3 = b3 ∗ x  +  c3 ∗ y  +  d3 ∗ z;
                             };
    public  abstract  a5  =  {
                                e1 = b1 ∗ x ∗ x  +  c1 ∗ y ∗ y;
                                e2 = b2 ∗ y ∗ y  +  c2 ∗ x ∗ x;
                             };
    public  SomeClass(){}
    public  void  setValues(...){...}
}
```

- And

- ◦ We
  - ▷ Execute:

$$\begin{array}{llcl} \text{SomeClass} & \text{sc} & = & \ldots; \\ \text{sc.setValues}(\ldots); & & & \\ [\text{float} \;\; \text{float} \;\; \text{float}] & \text{fl} & = & \text{sc.a4}[\text{x}, \;\; \text{y}, \;\; \text{z}]; \\ [\text{float} \;\; \text{float} \;\; \text{float} \;\; \text{float}] & \text{fl2} & = & \text{sc.a5}[\text{x}, \;\; \text{y}]; \\ [\text{float} \;\; \text{float}] & \text{fl3} & = & \text{sc.a4}[\text{y}, \;\; \text{x}]; \end{array}$$

there

- Will
  - ◦ Be
    - ▷ No change
      - – In:

$$\text{sc.x}, \qquad \text{sc.y} \qquad \textit{and} \qquad \text{sc.z},$$

- But:

$$\text{fl}[0][0], \qquad \text{fl}[0][1] \qquad \textit{and} \qquad \text{fl}[0][2]$$

will

- Contain
  - ◦ The
    - ▷ Values
      - – Of:

$$\text{x}, \qquad \text{y} \qquad \textit{and} \qquad \text{z}$$

after

- Solving
    - Those
        - Equations
            - In:

$$\text{sc.a4},$$

- And:

$$\text{fl.length} == 0,$$

if

- There
    - Is
        - No:
            - Solution,

- And:

$$\text{fl2[ ][0]} \qquad \textit{and} \qquad \text{fl2[ ][1]}$$

will

- Hold
    - The real
        - And imaginary
            - Parts of:

$$\text{x},$$

- And:

$$\text{fl2[ ][2, 3]}$$

that

- Of:

$$y.$$

- And
  - We
    - Can
      - Write:

$$[\text{float float}] \quad \text{fl4} \quad = \quad \text{a1}[\text{d1}, \ \text{d2}];$$

- Inside:

$$\text{SomeClass}.$$

- And:

$$\text{abstract} \quad \text{a6}, \quad \text{a7};$$

is

- Equivalent
  - To:

$$\text{abstract} \quad \text{a6} \quad = \quad \{\}, \quad \text{a7} \quad = \quad \{\};$$

- And:

$$\begin{array}{lllllllll}
\text{abstract} & & \text{a8} & = & \text{c1(i)} & :- & \ldots; & & \\
\text{a8} & & & |= & \text{c1("abc")}; & & & & \\
\text{abstract} & & \text{a9} & = & \text{b1} * \text{x} * \text{x} & - & \text{b2} & = & 0; \\
\text{abstract} & [\,] & \text{arr} & = & (& \{\ldots\}, & \{\ldots\} &); &
\end{array}$$

should

- Be
  - Rewritten
    - As:

```
abstract      a8   =   {  c1(i)   :−   ...;   };
a8                 |=  {  c1("abc");   };
abstract      a9   =   {  b1 ∗ x ∗ x  −  b2   =   0;   };
abstract      a10  =   {...},   a11   =   {...};
abstract [ ]  arr  =   (  a10,   a11   );
// And similarly, for trees and lists.
```

- And
  - We
    - Allow
      - Methods like:

```
abstract someMethod(abstract);
```

- And
  - We
    - Can
      - Write:

```
string  s              =   "abc";
native{
        <native-abstract-field>   |=   {   c1(s);   };
    }
```

in

- Static
    - Classes.